Hybrid Particle Swarm Optimiser with Breeding and Subpopulations

Morten Løvbjerg
EvALife project group
Dept. of Computer science
University of Aarhus
DK-8000 Aarhus C
Denmark
lovbjerg@daimi.au.dk
phone: +45 89423357

Thomas Kiel Rasmussen
EvALife project group
Dept. of Computer science
University of Aarhus
DK-8000 Aarhus C
Denmark
kiel@daimi.au.dk
phone: +45 89423357

Thiemo Krink
EvALife project group
Dept. of Computer science
University of Aarhus
DK-8000 Aarhus C
Denmark
krink@daimi.au.dk
phone: +45 89423358

Abstract

In this paper we present two hybrid Particle Swarm Optimisers combining the idea of the particle swarm with concepts from Evolutionary Algorithms. The hybrid PSOs combine the traditional velocity and position update rules with the ideas of breeding and subpopulations. Both hybrid models were tested and compared with the standard PSO and standard GA models. This is done to illustrate that PSOs with breeding strategies have the potential to achieve faster convergence and the potential to find a better solution. The objective of this paper is to describe how to make the hybrids benefit from genetic methods and to test their potential and competetiveness on function optimisation.

1 Introduction

The Particle Swarm Optimisation (PSO) algorithm was originally introduced in [Kennedy95] as an alternative to the standard Genetic Algorithm (GA). The PSO was inspired by insect swarms and has since proven to be a competitor to the standard GA when it comes to function optimisation. Since then several researchers have analysed the performance of the PSO with different settings, e.g., neighbourhood settings ([Kennedy99, Suganthan99]). Work presented in [Shi98] describes the complex task of parameter selection in the PSO model. Comparisons between PSOs and the standard GA were done analytically in [Eberhart98] and also with regards to performance in [Angeline98]. Angeline points out that the PSO performs well in the early iterations, but has problems reaching a near optimal solution in several real-valued function optimisation problems. Both Eberhart and Angeline conclude that hybrid models of the standard GA and the PSO, could lead to further advances.

We present such a hybrid model. The model incorporates one major aspect of the standard GA into the PSO, the reproduction. In the following we will refer to the used reproduction and recombination of genes only as "breeding". Breeding is one of the core elements that makes the standard GA a powerful algorithm. Hence our hypothesis was that a PSO hybrid with breeding has the potential to reach a better optimum than the standard PSO.

In addition to breeding we introduce a hybrid with both breeding and subpopulations. Subpopulations have previously been introduced to standard GA models mainly to prevent premature convergence to suboptimal points ([Spears94]). Our motivation for this extension was that the PSO models, including the hybrid PSO with breeding, also reach suboptimal solutions. Breeding between particles in different subpopulations was also added as an interaction mechanism between subpopulations.

The introduced hybrids were tested against both standard PSO and standard GA models.

The next section presents the structures of the hybrid PSO models. Section 3 describes the experimental settings used to find the results described in section 4. The experimental results are discussed in section 5 and finally section 6 summarises the study.

2 Model

The traditional PSO model, described by [Kennedy95], consists of a number of particles moving around in the search space, each representing a possible solution to a numerical problem. Each particle has a position vector ($\vec{x}_i$), a velocity vector ($\vec{v}_i$), the position ($\vec{p}_i$) and fitness of the best point encountered by the particle, and the index (g) of the

best particle in the swarm.

In each iteration the velocity of each particle is updated according to their best encountered position and the best position encountered by any particle, in the following way

$$\vec{v}_i = \chi(w\vec{v}_i + \vec{\varphi}_{1i}(\vec{p}_i - \vec{x}_i) + \vec{\varphi}_{2i}(\vec{p}_g - \vec{x}_i))$$

where χ is known as the *constriction coefficient* described in [Clerc99], w is the *inertia weight* described in [Shi98B, Shi98] and $\vec{p}_g$ is the best position known for all particles. φ_1 and φ_2 are random values different for each particle and for each dimension. If the velocity is higher than a certain limit, called V_{max}, this limit will be used as the new velocity for this particle in this dimension, thus keeping the particles within the search space.

The position of each particle is updated in each iteration. This is done by adding the velocity vector to the position vector, i.e.,

$$\vec{x}_i = \vec{x}_i + \vec{v}_i$$

The particles have no neighbourhood restrictions, meaning that each particle can affect all other particles. This neighbourhood is of type *star* (fully connected network), which have been shown to be a good neighbourhood type in [Kennedy99].

The structure of the hybrid model is illustrated in figure 1.

```
begin
    initialise
    while (not terminate-condition) do
        begin
            evaluate
            calculate new velocity vectors
            move
            breed
        end
end
```

Figure 1: The structure of the hybrid model.

The breeding is done by first determining which of the particles that should breed. This is done by iterating through all the particles and, with probability pb (*breeding probability*), mark a given particle for breeding. Note that the fitness is not used when selecting particles for breeding. From the pool of marked particles we now select two random particles for breeding. This is done until the pool of marked particles is empty. The parent particles are replaced by their offspring particles, thereby keeping the population size fixed.

The position of the offspring is found for each dimension by arithmetic crossover on the position of the parents, i.e.,

$$child_1(x_i) = p_i * parent_1(x_i) + (1.0 - p_i) * parent_2(x_i)$$

$$child_2(x_i) = p_i * parent_2(x_i) + (1.0 - p_i) * parent_1(x_i)$$

where p_i is a uniformly distributed random value between 0 and 1. The velocity vectors of the offspring is calculated as the sum of the velocity vectors of the parents normalised to the original length of each parent velocity vector.

$$child_1(\vec{v}) = \frac{parent_1(\vec{v}) + parent_2(\vec{v})}{|parent_1(\vec{v}) + parent_2(\vec{v})|}|parent_1(\vec{v})|$$

$$child_2(\vec{v}) = \frac{parent_1(\vec{v}) + parent_2(\vec{v})}{|parent_1(\vec{v}) + parent_2(\vec{v})|}|parent_2(\vec{v})|$$

The arithmetic crossover of positions and velocity vectors used were empirically tested to be the most promising. The arithmetic crossover of positions in the search space is one of the most commonly used crossover methods with standard real valued GAs, placing the offspring within the hypercube spanned by the parent particles. The main motivation behind the crossover is that offspring particles benefit from both parents. In theory this allows good examination of the search space between particles. Having two particles on different suboptimal peaks breed could result in an escape from a local optimum, and thus aid in achieving a better one.

We used the same idea for the crossover of the velocity vector. Adding the velocity vectors of the parents results in the velocity vector of the offspring. Thus each parent affects the direction of each offspring velocity vector equally. In order to control that the offspring velocity was not getting too fast or too slow, the offspring velocity vector is normalised to the length of the velocity vector of one of the parent particles.

Finally, the starting position of a new offspring particle is used as the initial value for this particle's best found optimum ($\vec{p}_i$).

2.1 Subpopulation Model

The motivation for introducing subpopulations is to restrict the gene flow (keeping the diversity) and thereby attempt to evade suboptimal convergence.

The subpopulation hybrid PSO model is an extension of the just described breeding hybrid PSO model. In this new model the particles are divided into a number of subpopulations. The purpose of the subpopulations is that each subpopulation has its own unique best known optimum. The velocity vector of a particle is updated as before except that the best known position ($\vec{p}_g$ in the formula) now refers to the best known position within the subpopulation that the particle belongs to. In terms of the neighbourhood topology suggested by Kennedy in [Kennedy99], each subpopulation has its own *star neighbourhood*.

The only interaction between subpopulations is if parents from different subpopulations breed. Breeding is now possible both within a subpopulation but also between different subpopulations. An extra parameter called *probability of same subpopulation breeding* (psb) determines whether a given particle selected for breeding is to breed within the same subpopulation (probability psb), or with a particle from another subpopulation (probability $1 - psb$).

Replacing each parent with an offspring particle ensures a constant subpopulation size.

3 Experimental Settings

Both the PSOs and the standard GA were tested on four benchmark problems, all minimisation problems. The first two functions were unimodal while the last two were multimodal with many local minima. All functions are designed such that their global minimum was at or near the origin of the search space.

The first test function was the generalised sphere function given by the equation

$$f_1(x) = \sum_{i=1}^{n} x_i^2$$

where x is a n dimensional real-valued vector and x_i is the ith element of that vector. The second function is the generalised Rosenbrock function given by the equation

$$f_2(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$

The third function is the generalised Griewank function.

$$f_3(x) = \frac{1}{4000} \sum_{i=1}^{n} (x_i - 100)^2 - \prod_{i=1}^{n} cos(\frac{x_i - 100}{\sqrt{i}}) + 1$$

The fourth and final test function is the generalised Rastrigin function which is given by the equation

$$f_4(x) = \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i) + 10)$$

These four functions have been commonly used in other studies on particle swarm optimisers (e.g. [Kennedy99, Shi99]).

The initial population is usually uniformly distributed over the entire search space. According to [Angeline98] this can give false indications of relative performance - especially if the search space is symmetric around the origin where many test functions have their global optimum. To prevent this, and to ease comparison with other models, the asymmetric initialisation method used in [Angeline98] was used.

Table 1: Search space and asymmetric initialisation ranges for each test function.

Function	Search space	Initialisation range
f_1	$-100 \leq x_i \leq 100$	$50 \leq x_i \leq 100$
f_2	$-100 \leq x_i \leq 100$	$15 \leq x_i \leq 30$
f_3	$-600 \leq x_i \leq 600$	$300 \leq x_i \leq 600$
f_4	$-10 \leq x_i \leq 10$	$2.56 \leq x_i \leq 5.12$

Search space and initialisation ranges for the experiments are listed in table 1. The number of generations run for each test function was set to 1000, 1500 and 2000 corresponding to the dimensions 10, 20 and 30 of the test functions respectively.

In both the standard PSO model and the hybrid model, the upper limits for φ_1 and φ_2 were set to 2.0, and a linearly decreasing inertia weight starting at 0.7 and ending at 0.4 was used. The constriction coefficient χ was set to 1. The maximum velocity (V_{max}) of each particle was set to be half the length of the search space in one dimension (for instance $V_{max} = 100$ for f_1 and f_2).

Two sets of experiments were conducted; Experiments with breeding alone and experiments with both breeding and subpopulations.

Research done in [Shi98] regarding scalability of the standard PSO have shown that the performance of the standard algorithm is not sensitive to the population size. Experiments with the hybrid model confirm this result. Based on these results the population size in the experiments was fixed to 20 particles in order to keep the computational requirements low.

In the experiments with subpopulations, the population size for the *whole* system was also 20. The size of each subpopulation was fixed throughout each run at $\frac{20}{subpopulations}$ particles.

The probability for breeding (pb) was empirically found to have its optimal setting at 0.2, which with 20 particles on average gives a total of two breedings per generation.

In the experiments with subpopulations, the best setting regarding the probability for breeding within the same subpopulation (psb) was determined empirically by examining the results for different settings. The number of subpopulations used in the experiments was 2, 3, 4 and 6. Table 2 shows the relation between the number of populations and the setting for this probability that appeared to be optimal.

The standard GA that we used was a real-valued GA with random initialisation, tournament selection with tournament size two, arithmetic crossover with random weight, Gaussian mutation with distribution $N(0, \alpha)$ where α is

Table 2: Probability for breeding within same subpopulation compared to number of populations

Populations	Psb
1	1.0
2	0.6
3	0.3
4	0.0
6	0.0

linearly decreasing from 1 to 0. Crossover and mutation probabilities for each of the four test functions are listed in table 3. In order to get a fair comparison between the models, with regards to the total number of evaluations, a population size of 20 individuals was also selected for the GA. This was done even though the standard GA often requires larger population sizes in comparison to the standard PSO model [Angeline98]. Other studies [Shi99] show that the standard PSO model with different population sizes have almost the same performance, so the low population size seems to be fair when analysing the PSO model.

Table 3: Crossover and mutation probability used in standard GA.

Function	Crossover prob.	Mutation prob.
f_1	0.60	0.30
f_2	0.50	0.30
f_3	0.50	0.40
f_4	0.20	0.02

A total of 100 runs for each experiment were conducted.

4 Experimental Results

Tables 4 and 5 list a representative set of results from the conducted experiments. The tables list the test function, the dimensionality of the function, the number of generations the algorithm was run and the average best fitness for the best particle found for the 100 runs of the four test functions respectively. Standard error for each value is also listed. Table 4 shows results for the experiments with the hybrid PSO without subpopulations. The table also list the corresponding average best fitness of both the standard PSO and the standard GA with the same settings (where they are applicable) as described in the previous section. Results for experiments with subpopulations are listed in table 5. Note that the hybrid PSO with one subpopulation in table 5 corresponds to the hybrid PSO in table 4.

Figures 2 to 7 are graphs corresponding to the reported experiments.

Figures 2 to 5 show the average best fitness for each generation for both the standard PSO model, the standard GA and the hybrid model. The graphs illustrate a representative set of experiments for functions with a dimensionality of 30. The hybrid model in these figures are without subpopulations (i.e. one subpopulation). Note that the figure with the Griewank function only illustrates two experiments, since the standard GA was unable to achieve a reasonable result (see table 4).

Figures 6 and 7 show the average best fitness for each generation for both the standard PSO model and the hybrid model. The graphs illustrate experiments with both a unimodal (Rosenbrock) and a multimodal test function (Griewank) both of 30 dimensions. The graphs for the hybrid model correspond to experiments with a varying number of subpopulations. The graphs for the standard PSO model are the same as in the previous figures.

Tables 4 and 5 with corresponding figures 2 to 5 show results for the standard PSO supporting the results in [Shi99].

In experiments with the Sphere function the standard PSO achieved better results and had much faster convergence than both the standard GA and the hybrid model with one subpopulation. The GA and the hybrid model found similar values but the hybrid model had a faster convergence speed than the GA. When the number of subpopulations in the hybrid model was increased the best fitness got worse. This happened in all of the experiments.

With the Rosenbrock function, the standard PSO had a better performance than both the GA and the hybrid model. The hybrid model only had a fitness comparable to that of the standard PSO when the test functions were of low dimensionality. When the dimensionality of the test functions were higher, the GA accomplished better results than the hybrid model. The convergence speed of the GA and the hybrid model was better than that of the standard PSO.

In the experiments with the Griewank function, the GA failed to achieve a reasonable result compared to the other models. The hybrid model had a faster convergence than the standard PSO, but achieved a marginally worse best value.

In experiments with the Rastrigin function, the hybrid model was better than both the standard GA and the standard PSO model with both a faster convergence and also a better best value found.

Table 4: Average best fitness of 100 runs for experiments without subpopulations (Average best fitness±standard error).

f	Dim.	Gen.	Std. PSO	Std. GA	Hybrid
f_1	10	1000	2.98E-33±4.21E-33	2.43E-04±1.14E-05	2.42E-04±2.17E-05
f_1	20	1500	3.03E-20±9.27E-21	0.00145±6.22E-05	0.00212±2.75E-04
f_1	30	2000	6.29E-13±7.64E-14	0.00442±1.78E-04	0.01203±6.33E-04
f_2	10	1000	43.049±11.554	109.810±6.212	43.521±16.047
f_2	20	1500	115.143±19.871	146.912±10.951	169.112±21.535
f_2	30	2000	154.519±24.512	199.730±16.285	187.033±22.960
f_3	10	1000	0.08976±0.00498	283.251±1.812	0.09078±0.03306
f_3	20	1500	0.03601±0.00298	611.266±3.572	0.00459±0.01209
f_3	30	2000	0.01504±0.00241	889.537±3.939	0.09911±0.00106
f_4	10	1000	4.8021±0.2323	3.1667±0.2237	3.0599±0.1535
f_4	20	1500	21.3917±0.7885	16.8732±0.6007	11.6590±0.3602
f_4	30	2000	46.9712±1.3206	49.3212±1.1204	27.8119±0.8059

Table 5: Average best fitness of 100 runs for experiments with subpopulations (Average best fitness±standard error). ("Hybrid (i)" is the hybrid model with i subpopulations).

f	Dim.	Gen.	**Hybrid** (1)	**Hybrid** (2)	**Hybrid** (4)	**Hybrid** (6)
f_1	10	1000	2.42E-04±2.17E-05	3.796E-05±9.22E-05	0.00223±9.13E-04	0.02124±0.00641
f_1	20	1500	0.00212±2.75E-04	0.00175±2.28E-04	0.00566±0.00185	0.04597±0.00721
f_1	30	2000	0.01203±6.33E-04	0.17396±4.56E-04	0.02023±0.00349	0.05669±0.00738
f_2	10	1000	43.521±16.047	51.701±13.761	63.369±14.006	81.283±14.907
f_2	20	1500	169.112±21.535	129.570±14.880	108.391±16.928	137.236±19.619
f_2	30	2000	187.033±22.960	196.554±14.733	279.390±19.468	247.724±31.822
f_3	10	1000	0.09078±0.03306	0.46423±0.03700	0.69206±0.02758	0.74694±0.01844
f_3	20	1500	0.00459±0.01209	0.02231±0.02121	0.09885±0.01883	0.34306±0.03072
f_3	30	2000	0.09911±0.00106	0.06316±0.00121	0.16389±0.00913	0.37501±0.02842
f_4	10	1000	3.0599±0.1535	3.5615±0.1478	3.6840±0.2611	6.8036±0.4657
f_4	20	1500	11.6590±0.3602	12.9158±0.3107	11.6379±0.5308	11.7054±0.5992
f_4	30	2000	27.8119±0.8059	38.5897±0.6455	29.5827±1.0649	29.1747±0.9449

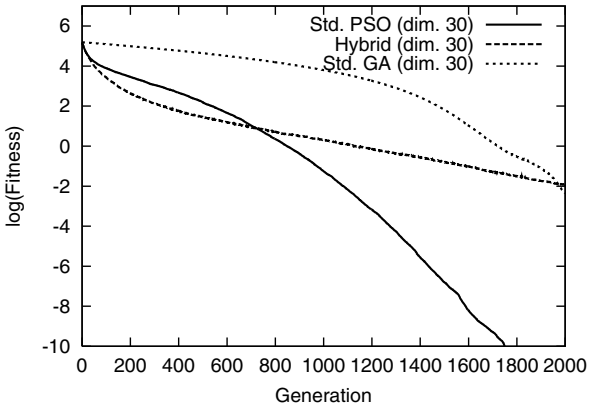


Figure 2: Standard PSO versus hybrid model for Sphere function with one population.

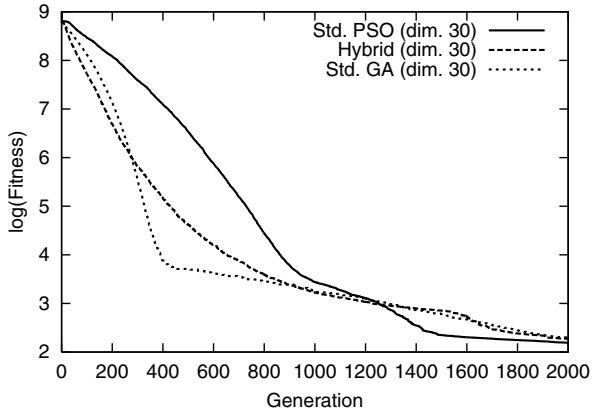


Figure 3: Standard PSO versus hybrid model for Rosenbrock function with one population.

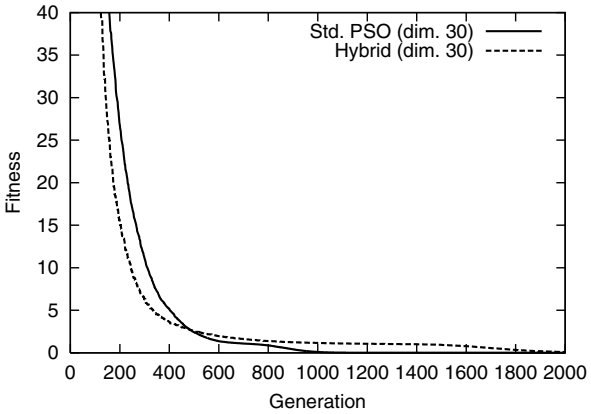


Figure 4: Standard PSO versus hybrid model for Griewank function with one population.

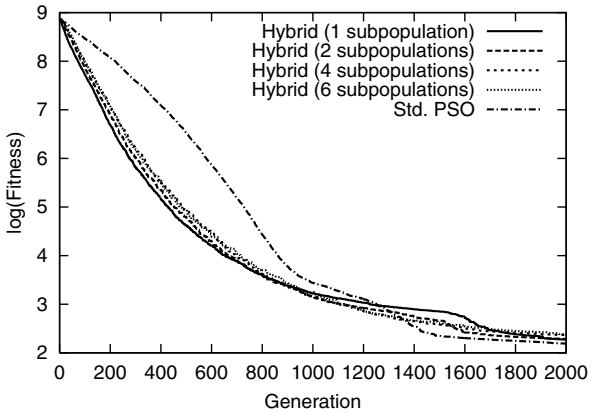


Figure 6: Hybrid model with different number of subpopulations versus standard PSO (Rosenbrock 30 dim.).

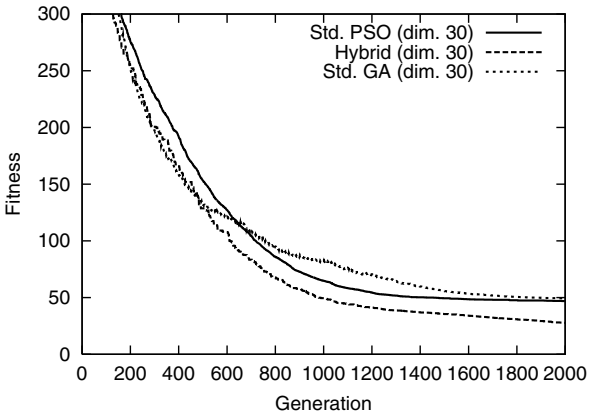


Figure 5: Standard PSO versus hybrid model for Rastrigin function with one population.

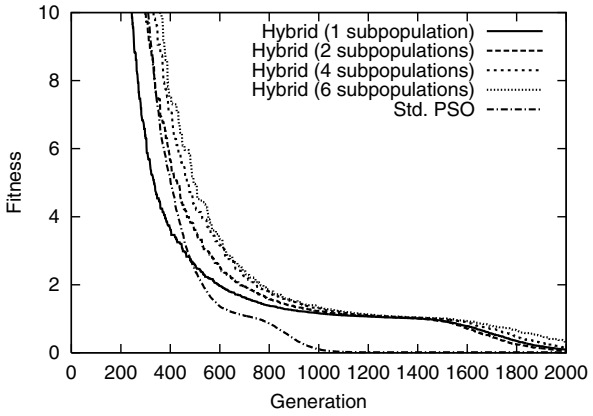


Figure 7: Hybrid model with different number of subpopulations versus standard PSO (Griewank 30 dim.).

5 Discussion

Tables 4 and 5 show a comparison of the performances in the standard PSO model, the standard GA, and the breeding PSO hybrid with regards to the optimum found.

Looking at the unimodal functions Sphere (f_1) and Rosenbrock (f_2) both the hybrid and the standard GA seem to outperform by the standard PSO. As mentioned in section 2 the offspring are initialised with a clean memory, i.e., the previously best found solution of a new particle is its starting point in the search space. This should provide a form of diversity since new particles are unaware of previously found optima. The purpose of adding diversity to the standard PSO is to tackle the problem of avoiding sub-optimal solutions. When we try to avoid sub-optimal solutions we run the risk of not beeing able to find a close to optimal solution because the particles takes longer to converge. This could be why the hybrid model suffers in experiments with unimodal functions.

Looking at the multimodal functions Griewank (f_3) and Rastrigin (f_4) the hybrid model should have a better chance of outperforming the standard PSO, because of the extra diversity. Table 4 does not show an improvement for the Griewank function, but figure 4 shows that the hybrid model converges faster than the standard PSO model. The standard GA was not able to reach a reasonable optimum in any of the experiments with the Griewank function. This is probably due to the fairly small population size in the GA. Table 4 along with figure 5 show the improvements for the Rastrigin function. Here both faster convergence is achieved and an improvement in the best solution is found. These results could be because of the design of the crossover operator that allows offspring particles to escape local optima (see section 2). The results seem to show

the potential of particle breeding regarding the multimodal problems.

Table 5 as well as figures 6 and 7 show no further increase in performance when subpopulations were introduced. Comparisons between the approach with one subpopulation (equal to the standard breeding PSO hybrid) and cases with more than one subpopulation show that the introduction of subpopulations only outperforms the standard breeding PSO hybrid in the Rosenbrock 20-dimensional function. In all other experiments the hybrid model with subpopulations performs worse than the standard PSO model. This is probably because the particles are distributed in several subpopulations which yields a subpopulation size that is too low.

The setting of psb, the probability of breeding within the same subpopulation, could be the cause of the performance deterioration. When the number of subpopulations is increased, the number of particles in each subpopulation is decreased. Having only a few particles in a subpopulation limits the effect of breeding within this subpopulation. Our experiments confirm that it was better to use a lower psb when the number of subpopulations increases, as seen in Table 2. A low psb implies that the probability for breeding between subpopulations is high which of course reduces the effect of subpopulations, in that the amount of gene flow in the total population is kept somewhat constant. These results suggest that the introduction of this specific subpopulation construction to the hybrid model does not generally improve the performance of particle swarms.

6 Conclusions and Future Work

In this paper a hybrid model based on the standard Particle Swarm Optimiser (PSO) and the standard Genetic Algorithm (GA) was introduced. The hybrid model was basically the standard PSO combined with arithmetic crossover. Furthermore, the notion of subpopulations in the hybrid model was introduced, also from the genetic algorithm field.

Four models were used in comparison, namely the standard PSO model, the standard GA and the two hybrid models. Parameters for each model were empirically tuned for each model yielding interesting results regarding the hybrid models. We found that the probability of breeding (pb) for a given particle had its optimum around 0.2. The optimal setting for the probability for breeding between subpopulations (psb) was in our case found to depend on the number of subpopulations. This result indicates that the model would work better with larger subpopulation sizes or other interaction constructions between subpopulations

On unimodal test functions (Sphere and Rosenbrock) the hybrid model was outperformed by the standard PSO and GA models regarding a comparison of the best optima found. Yet, the hybrid model had a marginally faster convergence than both the standard PSO and GA models. On multimodal test functions (Griewank and Rastrigin) the hybrid model performed better. The optima found by the hybrid were better or identical to those of the standard PSO model and the convergence speed was marginally faster.

Future work should cover the grounds of other subpopulation constructions. We chose breeding to model interaction between subpopulations, but other schemes such as migration should be investigated. Larger subpopulationsizes should also be investigated and compared to other evolutionary algorithms that uses subpopulations.

References

[Angeline98] P. J. Angeline, "Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences", Evolutionary Programming VII (1998), Lecture Notes in Computer Science 1447, 601–610. Springer.

[Clerc99] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization", Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, 1951–1957. IEEE Press.

[Eberhart98] R. C. Eberhart and Y. Shi, "Comparison between Genetic Algorithms and Particle Swarm Optimization", Evolutionary Programming VII (1998), Lecture Notes in Computer Science 1447, 611–616. Springer.

[Kennedy95] J. Kennedy and R. C. Eberhart, "Particle swarm optimization", Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, 1942–1948. IEEE Press.

[Kennedy99] J. Kennedy, "Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance", Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, 1931–1938. IEEE Press.

[Shi98] Y. Shi and R. C. Eberhart, "Parameter Selection in Particle Swarm Optimization", Evolutionary Programming VII (1998), Lecture Notes in Computer Science 1447, 591–600. Springer.

[Shi98B] Y. Shi and R. C. Eberhart, "A modified Particle Swarm Optimiser", IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9, 1998.

[Shi99] Y. Shi and R. C. Eberhart, ”Empirical Study of Particle Swarm Optimization”, Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, 1945–1950. IEEE Press.

[Spears94] W. M. Spears, ”Simple subpopulation schemes”, Proceedings of the Evolutionary Programming Conference 1994, pp. 296-307.

[Suganthan99] P. N. Suganthan, ”Particle Swarm Optimiser with Neighbourhood Operator”, Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, 1958–1962. IEEE Press.

Building Blocks: A Combinatorial Road To Greed

Anil Menon
Cerebellum Software
600 Waterfront Drive
Pittsburgh, PA 15222
anilm@acm.org

Abstract

The concept of building blocks is reconsidered in the context of set systems. It is argued that building block semantics can be captured by imposing two constraints (the weak heredity and weak augmentation properties). Building blocks that satisfy these properties are shown to be closely related (but not identical) to greedoids, combinatorial objects of central importance in the study of greedy algorithms.

1 Introduction

The aim of this paper is to examine the combinatorial structure of the building block concept. Two issues motivate this study: First, the need to clarify the controversial relationship between genetic algorithms and building blocks. Second, the need to study the (suspected) relationship between greedy algorithms and genetic algorithms.

The outline of this paper is as follows. In Section 2, the relationship between greedy algorithms and genetic algorithms (GAs) is considered. Section 3 examines the conceptual foundations of building blocks. It also introduces a representation to explicate the block interrelationships. Section 4 relates greedoids to a certain type of building blocks. Section 5 closes with a discussion of the results. Results drawn from external sources are referred to as "Propositions."

2 Greed & GAs: Connections

Besides the trivial fact that both greedy algorithms and genetic algorithms are algorithmic *strategies* rather than algorithms *per se*, they also share several other similarities:

- Both approaches involve the selection of some distinguished members from a population. This selection is not arbitrary, but instead is made with respect to some partial-order imposed on the population (usually, that induced by the cost function).
- Both require solutions to the problem to have certain structural properties, if optimal solutions are to be generated. These structural properties are particularly well understood in the case of greedy algorithms [7]. For genetic algorithms, the situation is much less clear. Minimally, one expects that there should be some correlation between the representation of a solution and its "goodness." Also, the early introduction of concepts like building blocks, deception, schemas, Royal Road functions etc. to the field, indicates the general acknowledgment of the importance of problem structure to GA-efficacy.
- Certain idealized versions of GAs can be shown to be gradient algorithms, which are merely greedy algorithms operating on cost surfaces. One such idealization is the GA equipped with proportional selection, no mutation effects, and applications of point crossover until linkage equilibrium is achieved in each crossover phase. This idealization is easily shown to be a gradient algorithm [13].
- Recent mathematical descriptions of the two approaches also bear strong resemblance to each other. The theory of greedy algorithms has been recast in terms of a new majorization operator acting on sequences [14]. Remarkably, an entirely different set of arguments (replicator theory, quadratic differential equations) enabled the interpretation of proportional selection, point crossover and bit mutation operators as majorization operators [11].

Of course, all this evidence is merely circumstantial, and some of it, defeasible. It is always harder to prove the absence of a relationship than the presence of one. Also, there are distinct differences between the two strategies. GAs are inherently probabilistic, while the "classic" applications of greedy algorithms (for example, the minimum spanning tree problem, Huffman encoding, the knapsack problem with real weights) are all inherently deterministic. The very notion of a probabilistically greedy algorithm is hard to define or operationalize. For instance, how would an "occasionally greedy" algorithm be different from a "randomly greedy" algorithm? Is a "non-greedy" algorithm allowed to be occasionally greedy?

There are other differences. GAs operate on a *population* of solutions, while greedy algorithms usually reduce to the incremental construction of an optimal object (for example, the construction of a minimum spanning tree, one edge at a time). Experimental studies with the Royal Road functions indicate that modifications that improve the performance of greedy algorithms do not necessarily improve the performance of genetic algorithms [2]. It is also doubtful that the two approaches are philosophically compatible. It is hard to see how diversity, that bedrock of evolution, can be achieved through purely greedy mechanisms.

Clearly, there are similarities as well as differences. But is there a conceptual core that is common to both approaches? The next two sections argue that there is such a core, rooted in the notion of a building block.

3 Reconsidering Building Blocks

The first explicit description of a building block was given by Goldberg:

> "Because highly fit schemata of low defining lengths and low order play such an important role in the action of genetic algorithms, we have already given them a special name: building blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high performance schemata, or building blocks." [3, pp. 41]

Intuitively, a building block is an allelic combination which confers upon its inheritor some attractive property, typically, an above-average fitness value. Lay explanations of GA dynamics have generally relied upon the notion of a building block. For example, deceptive functions are supposed to mislead the GA about the "right" building blocks to evolve, and crossover operators are sometimes ranked on their ability to combine building blocks. The concept of a building block has the virtues of being simple, reasonable, and useful in operator design.

The simplified explanation of how a GA works has come to be called the "building block hypothesis" (BBH). The BBH does not make any assertions on what should happen if low-fitness, low-order, schemas with low defining lengths are combined, nor does it concern itself with issues of statistical confounding. For these and related reasons, the BBH has been criticized [5, 12]. An influential paper showed experimentally that the BBH could not possibly be true in its lay interpretation [2]. Goldberg had introduced the concept of deception to encompass situations in which the BBH was violated; unfortunately, simulations of GAs optimizing deception-free functions (such as the Royal Road functions [2]) appeared to show that the BBH was still being violated. The concept of deception itself received a telling blow with Grefenstette's analysis [4].

However, the BBH and associated ideas can be defended in several ways. Altenberg's use of Price's theorem [1], or the recent attempt to relate Geiringer's theorem to schema analysis [15], shows how careful formulations of the Schema theorem can mitigate some of the criticisms leveled against it. The generality of Price's theorem makes it difficult to see how any evolutionary explanation can completely ignore schema-theoretic arguments. Besides, the BBH represents a certain ideal, that the nitty-gritty details of GA implementations may not achieve. This can be given a normative flavor: Would an optimizer really choose a GA that violates the BBH over one that preserves it?

Unfortunately, the BBH lends itself to misinterpretation. The assertion that a GA's success is due to the felicitous juxtaposition of building blocks is a causal assertion of the form "Y (high fitness schemata) *because of* X, (juxtaposition of building blocks)" or simply, "if X then Y." Seen this way, it suggests that a building block is a causal explanation in the sense of the philosopher John Mackie, namely, an INUS condition ("*I*nsufficient but *n*ecessary part of an *u*nnecessary but *s*ufficient condition[1]."") [10]. The BBH does not assert that highly fit schemata cannot be created in other ways (for example, random drift, combinatorial miracles via mutation, hitchiking mechanisms). It also says nothing about what would happen if there are constraints on the population size, that is, if the

[1] Italics added.

growth of one building block is at the expense of another. The case of cohort analysis in demographics is instructive. A wide variety of anomalous results may be demonstrated due to the dynamics of heterogeneous cohorts subject to different rates of growth [16]. Yet, mortality curves can still be constructed based on simple population models. The results of Forrest and Mitchell [2] appear to be closely related to the paradoxes observed when aggregation procedures are applied to heterogeneous collections [6].

Finally, it is important to remember that despite their common history, the BBH is distinct from the concept of a building block. The BBH may well be too simplistic for any practical or theoretical use. But the concept of a building block, or a variant thereof, is still a fruitful one.

In the next section, building blocks are examined from a combinatorial perspective. The idea is to represent a collection of building blocks as a set system, and translate the semantics of building blocks into structural statements on the set system. The resulting combinatorial structure will be compared with greedoids, and the similarities delineated.

3.1 Building Blocks: Representation

The first step is to fix the representational aspect of building blocks. Let Z_l denote the set of integers $\{1, 2, \ldots, l\}$. Assume that allelic values are drawn from the set Z_m. Let $P(Z_l \times Z_m)$ denote the power set (= set of all subsets) of $Z_l \times Z_m$. A building block is a set of the form,

$$B = \{(i,j) : i \in Z_l, j \in Z_m\} \in P(Z_l \times Z_m). \quad (1)$$

$(i,j) \in B$ indicates that the building block B has allele j in position i. Hence, it is required that if $(i_1,j) \in B$ and $(i_2,k) \in B$, then $i_1 \neq i_2$. The *order* of a building block B is its cardinality, that is, the number of elements in the set B, and is denoted as $|B|$. For example, the set $B = \{(1,1),(3,0),(5,1),(6,1)\}$ refers to the schema $1*0*11*\cdots$. Building blocks B_1 and B_2 are said to be in *conflict* if there exists a $(i,j) \in B_1$ and $(i,j') \in B_2$ and $j \neq j'$. Thus, the schemas $1*0*11$ and $0*0*01$ have two conflicting alleles at loci 1 and 5.

Let $\mathcal{B}(m,l)$ (or simply, $\mathcal{B}$) denote a set of building blocks on m alleles and l positions (loci).

The traditional interpretation of a building block as a high-fitness, low-order and low defining-length set of alleles is problematic for three reasons. First, the defining length of a building block is an artifact of the "string" representation (unlike its order), and may not be meaningful for other representations. Second, the term "high fitness" can be interpreted in several non-equivalent ways ("above average fitness", "above median-fitness" etc.). Third, it is not clear whether a building block is a static concept or a dynamic one (that is, proportion dependent).

The view adopted in this paper is as follows: The defining length of a building block will be treated as not being relevant to the concept of a building block. It is assumed that the building block concept is a static one, and hence should not depend on the schema proportions in the population. Finally, by "high fitness" it is meant that for the non-negative, real valued, fitness function F under consideration, there is some fitness-criteria based, many-to-one indicator function μ_F, where:

$$\begin{aligned} \mu_F : P(Z_l \times Z_m) &\rightarrow \{0,1\}, \\ B_i \in \mathcal{B} &\Leftrightarrow \mu_F(B_i) = 1. \end{aligned}$$

For example, the indicator function could be based on whether the fitness of a block was greater than the static average of the function. The function $\mu_F(\cdot)$ is deliberately left unspecified, because the choice of a specific formula is not relevant as long as it is used consistently (at least for the purposes of this paper).

3.2 Encodons: Encoding Building Blocks

Each building block in $\mathcal{B}(m,l)$ is now *encoded* as a subset of Z_q where $q \leq |l^m|$. The encoding will be represented by the one-to-one *partial* function η:

$$\begin{aligned} \eta : P(Z_l \times Z_m) &\rightarrow Z_{l^m}, \\ \eta(\phi) &= \phi. \end{aligned}$$

ϕ is said to be the *trivial* encodon. The partial function η is computed by the following procedure, formally described below in Figure 1.

The algorithm in Figure 1 operates (roughly) like a variable length encoding procedure. It begins by renumbering the elements of $\mathcal{B}$ so that the first l_1 (say) elements are all blocks of order 1, the next l_2 elements are of order 2 and so on [step 1]. Encode each order-d building block in terms of the encodings of blocks of lowest possible order [step 2].

For example, suppose a block of order-d can be written as the union of a collection of order-1 blocks. Its encoding is then defined to be the union of the encodings of the order-1 blocks in the collection. However, an order-d building block need not usually decompose in that manner. The general method for encoding such a block X is to determine its intersection with

1. Rearrange the elements of $\mathcal{B}$ so that the block orders are non-decreasing. Set $cnt = 0$.
2. For each successive order-d block B_j in $\mathcal{B}$, do:
   (a) Define $\eta(B_j) = \phi$.
   (b) For each order-r block B_k ($k \neq j, r < d$) in $\mathcal{B}$ do:
      i. Compute $C_{jk} = B_j \cap B_k$.
      ii. If $\eta(C_{jk})$ is *not* defined, then: define $\eta(C_{jk}) = \{cnt+1\}$. $cnt = cnt+1$.
      iii. Update: $\eta(B_j) = \eta(B_j) \cup \eta(C_{jk})$.
   (c) Compute $D_j = B_j - \cup_{r \in \eta(B_j)} B_r$.
   (d) If $D_j \neq \phi$, then:
      i. Define $\eta(D_j) = \{cnt+1\}$. $cnt = cnt+1$.
      ii. Set $\eta(B_j) = \eta(B_j) \cup \eta(D_j)$.

Figure 1: Computing Block Encodings

every other building block of strictly lower order [step 2(b)i]. If an intersection set has already been encoded, then its encoding is added to X's encoding set [step 2(b)iii]. If not, we define the intersection set's encoding to be a new singleton set consisting of the smallest unused integer in Z_{l^m} [step 2(b)ii]. That encoding is also added to X's encoding set [step 2(b)iii]. If X can be recovered by taking the union of its intersections with other blocks, then we are done encoding X [step 2(d)]. Else, X consists of a portion (set) that does not intersect with any other block. That portion is encoded as a unique singleton set consisting of the smallest unused number in Z_{l^m} [step 2(d)i], and the encoding also added to that of X [step 2(d)ii]. At the end of this procedure each building block in $\mathcal{B}(l,m)$ will be represented uniquely as some subset of Z_q for some $q \leq |l^m|$.

For example, using this procedure, one encoding for the building blocks associated with the Royal Road function $R1$ could be: $s_i \rightarrow \{i\}$ (for $i = 1 \ldots 7$) and $s_8 \rightarrow \{1,2,3,4,5,6,7\}$. Similarly, the building blocks in the Royal Road function $R2$ could be mapped as $s_i \rightarrow \{i\}$ (for $i = 1, \ldots, 8$), $s_9 \rightarrow \{1,2\}$,$s_{10} \rightarrow \{3,4\}$, $s_{11} \rightarrow \{5,6\}$, $s_{12} \rightarrow \{7,8\}$ and $s_{13} \rightarrow \{1,2,3,4,5,6,7,8\}$.

The encoding of a building block (that is, the image of the building block under η) will be referred to as its **encodon**. The *length* of an encodon is its cardinality, and is *not* related in an obvious way to the order of the building blocks. Define the set of encodons $\mathcal{C}$ by:

$$\mathcal{C} = \eta(\mathcal{B}) = \{\eta(B_j) : B_j \in \mathcal{B}). \quad (2)$$

Two encodons are said to be *conflicting* if they are images of conflicting building blocks (section 3.1).

The set of encodons corresponding to a set of blocks are unique upto permutation (briefly, this is because the encodons record building block intersections). Reordering the order-k blocks differently, could result in a different subset assignment. It is possible to modify the algorithm in Figure 1 to get a unique set of codons, but the results of this paper only need uniqueness upto permutation.

In the next section, a subset of building blocks are considered; one that will clarify the relationship between greedy algorithms and building blocks.

4 Matroyshka Blocks and Greedoids

What makes encodons something more than schemas in formal wear, is the definition of a Matroyshka set.

Definition 1 (**Matryoshka**[2] **Blocks**): A set of building blocks, $\mathcal{B}$ is said to be a *Matroyshka* set, if its corresponding encodon set $\mathcal{C} = \eta(\mathcal{B})$ satisfies the following two properties:

(**Ground property**): $\phi \in \mathcal{C}$.

(**Weak Heredity property**): An encodon of length $k > 1$ must contain at least one non-trivial encodon. Formally, if $C \in \mathcal{C}$ and $|C| > 1$, then there exists a proper subset $C' \subset C$, $|C| > |C'| > 0$ and $C' \in \mathcal{C}$. ∎

The Ground property is added to ensure that encodons of size 1 will not trivially violate the Weak Heredity property. It does no harm, and simplies proofs.

The basic reason why Matryoshka sets are defined with respect to encodons, and not building blocks, is that the encodons enable the building blocks *to be treated as blocks*. For example, had the building blocks been required to satisfy Weak Heredity (instead of their encodons), then the Royal Road functions would appear not to satisfy it. But the functions were explicitly designed to have building blocks made up of lower order blocks. That explicit construction is visible when we look at the intersections between the building blocks, that is, their encodons (roughly).

[2]The name was inspired by the Matroyshka dolls; these are "nested dolls," a traditional Russian folkcraft.

The Weak Heredity property requires that an encodon must contain at least one other encodon of non-zero length. Note that just *one* such 'sub-encodon" is required. The encodons obtained from the Royal Road functions $R1$ and $R2$ do have this property. But it is easy to construct scenarios where this property is not achieved. For example, suppose $\mathcal{B}$ was specified by the schemas $\{**\ *, 00*, 0*0, *00\}$, corresponding to the descriptions, $\mathcal{B} = \{B_1, B_2, B_3, B_4\}$, with, $B_1 = \phi$, $B_2 = \{(1,0),(2,0)\}$, $B_3 = \{(1,0),(3,0)\}$, $B_4 = \{(2,0),(3,0)\}$. One possible encoding of $\mathcal{B}$ is given by, $\eta(B_1) = \phi$, $\eta(B_4) = \{1,2\}$, $\eta(B_5) = \{1,3\}$ and $\eta(B_6) = \{2,3\}$. $\mathcal{B}$ is not a Matroyshka set because $\eta(B_4)$, $\eta(B_5)$, and $\eta(B_6)$ break the Weak Heredity property.

The first argument for Weak Heredity is that it defines a certain kind of "consistency." Suppose a set of alleles was defined to be a building block if its static fitness (averaged over the fitnesses of its instances) was greater than or equal to the average value of the function. Suppose further that a schema of order d, say, $1*00*$ happened to be a building block, but none of its "sub-schemas" (for example, $**00*$) was a building block. That is, the fitnesses of each of the non-conflicting sub-schemas are below the average fitness. This is a *deceptive* situation. A lower order schema gives information conflicting with a higher order schema over the same set of fixed alleles. The idea behind requiring a set of encodons to possess the Weak Heredity property is to prevent this kind of occurrence. When one encodon contains another it indicates that a subset of alleles consistently manifests in a lower order building block as well as a higher order one. This scenario is only mildly dependent on characterizing a building block as having "higher than average fitness." Other centralized measures (median, mean, mode et cetera) could be used, and the thrust of the argument would not be weakened.

A second argument is that the Weak Heredity property models the reasoning behind the construction of functions like the Royal Road functions. The property (theoretically) would allow the generation of good solutions from previous ones, with a minimal reliance on combinatorial miracles. As building blocks are currently understood, even delta functions could be said to "possess" building blocks. For example, in the function defined by $f(111) = 100$, and otherwise $f(***) = 0$, the block 111 could be said to be a "building block." To allow such extreme scenarios is to dilute the concept of a building block. If knowing that a function possesses a set of building blocks includes the possibility that the blocks are merely a series of disconnected "needles in the haystack,", then such knowledge is of very limited use. Irrespective of whether or not GAs (or any other algorithmic strategy) actually take advantage of such a property, Weak Heredity can be used to develop a performance ideal, against which actual performance can be compared.

A third argument is that the Weak Heredity property is implicit in the notion of "building" from building blocks. To see this, consider the following property.

(**Weak Augmentation**): Let $C_1, C_2 \in \mathcal{C}$ denote two non-conflicting encodons, such that $|C_1| > |C_2|$. Then, there exists a non-empty set $X \subseteq C_1 - C_2$, such that $|X| < |C_1 \cup C_2|$ and $C_2 \cup X \in \mathcal{C}$. ∎

The Weak Augmentation property says that given two building blocks, one larger than the other, and not sharing any conflicting alleles, there must be *some* alleles in the difference of the two pieces, such that the addition of those alleles to the smaller building block, produces a new building block. The requirement of non-conflict is a way of saying that either the building blocks agree on their common alleles or they are specified over disjoint sets of alleles. It is a technicality to prevent meaningless or undefined mixes of encodon sets.

The idea of "building" in building blocks is thus modeled by requiring that some subset may be transferred from the larger block to the smaller one without losing quality. Typically, one visualized the word "building" as the "putting together" of units. This definition on the other hand, views "building" in terms of the relation between the end product and one of its constituent pieces. The intuitive meaning of the word "building" is not lost; see for example, Proposition 1.

What makes the Weak Augmentation property particularly interesting is that encodons that satisfy it also possess the Weak Heredity property, as is shown by the following lemma.

Lemma 1 If a set of encodons $\mathcal{C}$ satisfies the Ground and Weak Augmentation properties, then it also satisfies the Weak Heredity property. ∎

Proof: Satisfaction of the Ground property implies that $\phi \in \mathcal{C}$. Let $C_i \in \mathcal{C}$ be any encodon such that $|C_i| \geq 2$ (If there is not any such encodon then the Weak Augmentation property is trivially satisfied and the proof is done). By definition, C_i is not in conflict with the trivial encodon ϕ. Applying the Weak Augmentation to the pair (C_i, ϕ), implies the existence of a non-empty $C_i' \subset C_i$ such that $C_i' \in \mathcal{C}$. But C_i is any encodon in $\mathcal{C}$ (of non-unit size), and the lemma is proved. ∎

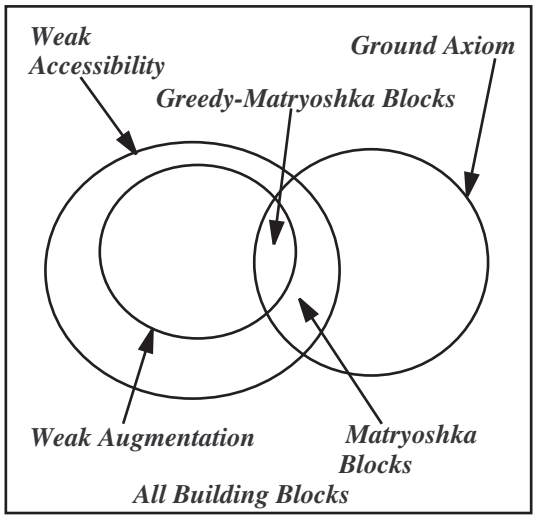


Figure 2: Building Block Categories

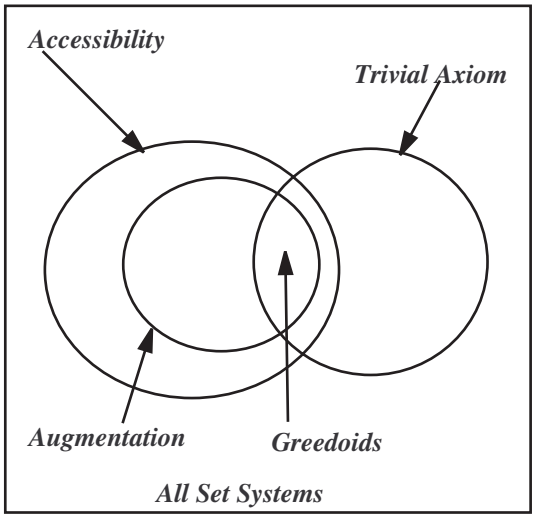


Figure 3: Set Structure of Greedoids

If a set of building blocks satisfies both the Weak Augmentation and the Ground properties, then it said to belong to the family of *Greedy-Matryoshka* set (the reason for the name will become clear shortly). From Lemma 1 it follows that if a set of blocks is Greedy-Matryoshka then it is also a Matryoshka set

The adjective "weak" is appended to the heredity and augmentation properties because each of these properties has a strong form. The "strong" version of the Weak Augmentation property is given below.

(**Augmentation**): Let C_1 and C_2 denote two non-conflicting encodons such that $|C_1| > |C_2|$ and $|C_1| > 0$. Then, there exists an *element* $x \in C_1 - C_2$ such that $C_2 \cup \{x\} \in \mathcal{C}$. ∎

The Augmentation property is much more specific than the corresponding weak version as to the size of the exchange required to generate a new encodon from the original pair. A corresponding "strong" version of the Heredity property can also be defined.

(**Heredity**): An encodon of length $k > 1$ must contain at least one non-trivial encodon of length $k - 1$. Formally, if $C \in \mathcal{C}$, and $|C| > 1$, then there exists a proper subset $C' \subset C$, $|C| > |C'| > 0$ and $C' \in \mathcal{C}$. ∎

Lemma 1 derived Weak Heredity from the Ground and Weak Augmentation properties. In an analogous manner, it can be shown if a set of encodons satisfies the Augmentation and Ground properties, then it also satisfies the Heredity property. In other words:

Lemma 2: If a set of encodons $\mathcal{C}$ satisfies the Ground and Augmentation properties, then it also satisfies the Heredity property. ∎

In fact, given that the Ground property holds, it can be shown that if just one of the properties (either augmentation or heredity) is in the "strong" version, then the resulting system of encodons has the other property in the strong form as well. It suffices therefore, to consider encodon sets with either (a) both properties in the strong version, (b) both properties in the weak version, (c) possessing only the Weak Heredity property (Matryoshka sets) or (d) not possessing even the Weak Heredity property. The relationship of the various block sets is shown in Figure 2.

The concept of a Matryoshka block can now be connected to the theory of greedoids [8], and related set systems [7].

Definition 2 (**Greedoids**): Let F be some collection of subsets of a finite set E, that is, $F \subseteq P(E)$, where $P(E)$ denotes the power set of E. The tuple $G = (E, F)$ is said to be a *greedoid* if the following two axioms are satisfied:

(*Trivial Axiom*): $\phi \in F$, i.e. the empty set belongs to F.
(*Augmentation Axiom*): If $X, Y \in F$ and $|X| > |Y|$ then there exists an $x \in X - Y$ such that $Y \cup \{x\} \in F$.

E will be called the *ground set*. ∎

Results analogous to Lemma 1 exist in the theory of greedoids as well. For example, it can be shown that the Trivial axiom and the Augmentation axiom imply the following property:

(**Accessibility**) For all $X \in F$ there exists an $x \in X$ such that $X - \{x\} \in F$. ∎

The Accessibility property is essentially identical to the (strong) Heredity property. The latter name is retained on account of its suitability in the evolutionary context.

Upon comparing Definition 2 with that of a Matroyshka set (Figures 2 and 3 depict the similarities visually) it is clear that greedoids are a "stronger" version of the idea of Matryoshka blocks. From a combinatorial perspective, the difference between the two is one of *degree*, rather than *quality*. Since the notion of a Matryoshka set was motivated by building block semantics, the connection with greedoids is interesting.

This similarity finds reflection in several results in the greedoids literature. For example, consider the following proposition, known (oddly enough) in the greedoid literature as the *recombination lemma* [8, Lemma 4.2].

Proposition 1 (Recombination Lemma) Let (E, F) be a greedoid, and $X \cup Y \in F$ and $U \in F$. If $|U| = |X|$, then $U \cup Y \in F$. ∎

Notice the resemblance to the operating philosophy of the point crossover operators. The existence of such results in the greedoids literature and the fact that Matryoshka blocks are (approximately) weakened greedoids, implies that many of those results also carry over in a weakened form. For example, the concept of rank is fundamental in the study of set systems. It can be applied to Greedy-Matryoshka sets as follows.

Let $\mathcal{B}(m, l)$ be a set of building blocks, and $\mathcal{C} = \eta(\mathcal{B})$ be a Greedy-Matryoshka set. Define $E = \cup_{i:\, C_i \in \mathcal{C}} C_i$. The (independence) *rank* of a set $X \subseteq E$ with respect to $\mathcal{C}$ is defined as:

$$\rho(X) = \max\{|A| : A \subseteq X, A \in \mathcal{C}\}. \qquad (3)$$

With the definition of rank in hand, Lemma 3 characterizes a Greedy-Matryoshka set in terms of the rank function.

Lemma 3: A function $\rho : 2^E \rightarrow Z$ is[3] the rank function of a Greedy-Matryoshka set if and only if for all $X, Y \subseteq E$ and $x, y \in E$:

(R1) $\rho(\phi) = 0$,
(R2) $\rho(X) = |X|$,
(R3) If $X \subseteq Y$ then $\rho(X) \leq \rho(Y)$,
(R4) If $\rho(X) = \rho(X \cup \{x\}) = \rho(X \cup \{y\})$, then $\rho(X) \leq \rho(X \cup \{x\} \cup \{y\})$.

Furthermore, the rank function determines the greedy-Matryoshka set uniquely. ∎

[3] Z is the set of non-negative integers.

Proof: The proof is almost identical to the one for Theorem 2.3 in [8] and will not be duplicated.

One course of action is to define the concept of a "base" (elements of maximal rank in a set of encodons) as in greedoids, and then define optimization problems on the set of bases of a greedoid. Many optimization problems can be so represented. The advantage of such a specification is that it can be used to derive time and space complexity results. But rather than pursue weakened versions of known results in Greedoid theory, it is more profitable to clarify the relationship between greedy algorithms and GAs.

5 Discussion

The arguments of the last section do *not* imply that GAs are nothing more than variants of greedy algorithms. The previous section argued for a class of building blocks that are weakened versions of greedoids. It does not claim that GAs will be good, bad or indifferent for optimization functions defined on these set systems. GA performance cannot be inferred for the simple reason that GA dynamics has not been taken into account. The analysis is combinatorial and axiomatic, rather than behavioral and inductive. GA dynamics can perhaps be studied from a "greedy" viewpoint using majorization theory, but that is not the focus of this paper.

The second point is that greedoids do not *completely* characterize the class of greedy algorithms. Specifically, there are greedy algorithms that do not have underlying greedoids (and hence, greedoids are too restrictive), and there are greedy algorithms that do not return an optimal solution when run on a greedoid (thus, greedoids are too general). This vexing situation has been resolved, and the class of functions for which a "greedy algorithm" produces the optimal solution has now been completely characterized[4] [7]. In other words just because a subset of building blocks happen to formally related to greedoids is not grounds for concluding that GAs are greedy algorithms. Finally, algorithmic efficiency (time complexity for finding an optimal solution) differs from algorithmic sufficiency (ability to find an optimal solution)[5].

What *can* be claimed is that both greedy algorithms as

[4] This class of functions is quite general, non-trivial, and defined in terms of *matroid embeddings*, an extension of the concept of greedoids.

[5] Greedy $\not\Rightarrow$ "easy." A greedy algorithm may be guaranteed to find an optimal solution for a particular problem, but may do so very inefficiently. There are also problems that while solvable by a greedy solution, are also NP-hard [9, Section 6].

well as genetic algorithms have a common conceptual core. This core, consisting of certain kinds of set systems, is useful for clarifying discussions, characterizing algorithmic sufficiency, and providing a combinatorial basis for evolutionary algorithms.

One can also speculate that the connection with greedy algorithms reveals the need for a theory of "probabilistic greed." Probabilistic versions of deterministic algorithms often have capabilities not possessed by the latter (e.g. primality testing, simulated annealing), even to the extent of belonging to different complexity classes. If the conceptual difficulties associated with the notion of probabilistic greed (as briefly discussed in Section 2) are overcome, then the kind of precise results obtained for simulated annealing or approximate exact sampling, may be within reach. In the final analysis, what counts is the use one makes of these ideas to build better genetic algorithms. The ideas outlined here are meant to contribute towards efforts in that direction.

Acknowledgments

I would like to thank Dennis Wakefield for his suggestions, and for patiently enduring various iterations of the document. Lera Brusilovsky was kind enough to instruct me on the correct usage of the word "Matryoshka."

References

[1] L. Altenberg. The Schema theorem and Price's theorem. In D. Whitley and M. Vose, editors, *Foundations of Genetic Algorithms*, volume 3, pages 23–50. Morgan Kaufmann, 1994.

[2] S. Forrest and M. Mitchell. Relative building block fitness and the building-block hypothesis. In D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 109–126. Morgan Kaufmann, 1992.

[3] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[4] J. Grefenstette. Deception considered harmful. In D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 75–92. Morgan Kaufmann, 1993.

[5] J. Grefenstette and J. Baker. How genetic algorithms work: a critical look at implicit parallelism. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1. Morgan Kaufmann, 1989.

[6] D. B. Haunsperger and D. G. Saari. The lack of consistency for statistical decision procedures. *The American Statistician*, 45(3):252–255, 1991.

[7] P. Helman, B. M. E. Moret, and H. D. Shapiro. An exact characterization of greedy structures. *SIAM J. Discrete Mathematics*, 6(2):274–283, 1993.

[8] B. Korte and L. Lovász. Greedoids: A structural framework for the greedy algorithm. In W. R. Pulleybank, editor, *Progress in Combinatorial Optimization*, pages 221–243. Academic Press, New York, 1984.

[9] B. Korte and L. Lovász. Greedoids and linear objective functions. *SIAM J. of Algebraic and Discrete Methods*, 5(2):229–238, 1984.

[10] J. L. Mackie. Causes and conditions. In E. Sosa and M. Tooley, editors, *Causation*, Oxford Readings in Philosophy, pages 33–55. Oxford University Press, N.Y., 1993.

[11] A. Menon, K. Mehrotra, C. Mohan, and S. Ranka. Replicators, majorization and genetic algorithms: New models, connections and analytical tools. In R. Belew and M. Vose, editors, *Foundations of Genetic Algorithms*, volume 4, pages 155–180. Morgan Kaufman, 1997.

[12] H. Mühlenbein. Evolution in time and space — the parallel genetic algorithm. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 316–338. Morgan Kaufmann, 1991.

[13] H. Mühlenbein. The equation for the response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1998.

[14] Prasad Ram. *A new understanding of greed*. PhD thesis, Dept. of Computer Science, University of California, Los Angeles, 1993. online: citeseer.nj.nec.com/ram93new.html.

[15] C. Stephens and H. Waelbroeck. Schemata evolution and building blocks. *Evolutionary Computation*, 7(2):109–124, 1999.

[16] J. W. Vaupel and A. I. Yashin. Heterogenity's ruses: some surprising effects of selection on population dynamics. *The American Statistician*, 39(3):176–185, 1985.

Three-Objective Genetic Algorithms for Designing Compact Fuzzy Rule-Based Systems for Pattern Classification Problems

Tadahiko Murata, **Shuhei Kawakami**, **Hiroyuki Nozawa**, **Mitsuo Gen**
Department of Industrial and Information Systems Engineering
Ashikaga Institute of Technology
268 Omae-cho, Ashikaga, 326-8558, Japan

Hisao Ishibuchi
Department of Industrial Engineering
Osaka Prefecture University
1-1 Gakuen-cho, Sakai, 599-8531, Japan

Abstract

In this paper, we formulate the design of fuzzy rule-based classification systems as a three-objective optimization problem. Three objectives are to maximize the classification performance of a fuzzy rule-based system, to minimize the number of fuzzy rules, and to minimize the number of features used in the fuzzy rule-based system (i.e., used in the antecedent part of fuzzy rules). The second and third objectives are related to simplicity and comprehensibility of the fuzzy rule-based system. We describe and compare two genetic-algorithm-based approaches for finding non-dominated solutions (i.e., non-dominated fuzzy rule-based systems) with respect to the three objectives. One approach is a rule selection method where a small number of linguistic rules are selected from prespecified candidate rules by a genetic algorithm. The other is a fuzzy partition method, which designs fuzzy rule-based systems by simultaneously determining the number and the shape of the membership function of each fuzzy set from training patterns. These two approaches are compared with each other through computer simulations on some real-world classification problems such as iris data, wine data, and glass data.

1 INTRODUCTION

Genetic algorithms have been successfully applied to various optimization problems (Goldberg 1989). The extension of GAs to multi-objective optimization was proposed in several manners (Schaffer 1985, Kursawe 1991, Horn 1994, Fonseca 1995, Murata 1995, Zitzler 1999). The aim of these algorithms is to find a set of Pareto-optimal solutions of a multi-objective optimization problem. Another issue in multi-objective optimization is to select a single final solution from Pareto-optimal solutions. Many studies on multi-objective GAs did not address this issue because the selection totally depends on the decision maker's preference. In this paper, we also concentrate our attention on the search for finding a set of Pareto-optimal solutions. We apply a multi-objective genetic algorithm to classification problems for constructing classification systems.

Our task in this paper is to design comprehensible fuzzy rule-based systems for high-dimensional pattern classification problems. Recently, some researchers (Pedrycz 1996, Setnes 1998a, 1998b, 2000, Yen 1998, 1999, Jin 1999, 2000, and Oliveira 1999) tried to improve interpretability of fuzzy rule-based systems. For example, interpretability of membership functions was discussed in (Pedrycz 1996, Oliveira 1999). The number of fuzzy rules was decreased in (Setnes 1998a, 1998b, 2000, and Yen 1999). Jin (2000) pointed out the following four factors closely related to interpretability of fuzzy rule-based systems.

(a) Distinguishability of a fuzzy partition. Membership functions should be clearly distinguishable from each other so that a linguistic term can be assigned to each membership function.
(b) Consistency of fuzzy rules. Fuzzy rules in a fuzzy rule-based system should not be strongly contradictory to each other.
(c) The number of fuzzy rules. It is easy to examine a small number of fuzzy rules while the examination of many rules is a cumbersome task.
(d) The number of conditions in the antecedent part (i.e., if-part). It is not easy to understand a fuzzy rule with many antecedent conditions.

Among these four factors, distinguishability was included in a cost function in regularized learning of Jin (2000).

In this paper, we describe and compare two genetic-algorithm-based approaches for finding non-dominated solutions (i.e., non-dominated fuzzy rule-based systems) with respect to the three objectives. One approach is a rule selection method (Ishibuch *et al.* 1997) where a small number of linguistic rules are selected from prespecified candidate rules by a genetic algorithm. The other approach is a fuzzy partition method,

which designs fuzzy rule-based systems by simultaneously determining the number and the shape of the membership function of each fuzzy set from training patterns. Both the approaches have been already proposed as single-objective genetic-algorithm-based approaches. We apply a multi-objective genetic algorithm to both the approaches in this paper.

As for the first interpretability factor (a), we do not have to consider the distinguishability of a fuzzy partition in the rule selection method, since we use prespecified linguistic terms with fixed membership functions. On the other hand, we should carefully examine constructed classification systems by the fuzzy partition method, since the number and the shape of membership functions vary during the execution of the method. The consistency of fuzzy rules (i.e., the second factor (b)) is resolved by assigning a certainty grade to each fuzzy rule in the two approaches.

In order to consider the last two factors (c) and (d), we formulate the design of fuzzy rule-based classification systems as a three-objective optimization problem. Three objectives are to maximize the classification performance of a fuzzy rule-based system, to minimize the number of fuzzy rules, and to minimize the number of features used in the fuzzy rule-based system (i.e., used in the antecedent part of fuzzy rules). The second and the third objectives show that (c) and (d) are considered in our optimization problem.

Since both the approaches are fuzzy rule-based systems, we firstly explain fuzzy rule-based system employed in this paper. We show the main difference between the two genetic-algorithm-based approaches in this section. Then we show the common architecture in the multi-objective genetic algorithm employed for both the approaches. Next, we show genetic operations in them. Finally, these two approaches are compared with each other through computer simulations on some real-world classification problems such as iris data, wine data, and glass data.

2 FUZZY RULE-BASED SYSTEMS

We assume that *m* training patterns (i.e., labeled patterns) are given as numerical data for an *n*-dimensional *c*-class pattern classification problem. We denote those training patterns as $\mathbf{x} = (x_{p1},...,x_{pn})$, $p = 1,2,...,m$. For simplicity of explanation, each attribute value x_{pi} is assumed to be a real number in the unit interval [0, 1], i.e., $x_{pi} \in [0, 1]$. This means that the pattern space of our pattern classification problem is the *n*-dimensional unit hypercube $[0, 1]^n$. In computer simulations of this paper, all attribute values are normalized into real numbers in the unit interval [0, 1]. For an *n*-dimensional and *c*-class pattern classification problem, we try to find fuzzy rules of the following form:

$$\text{Rule } R_j: \text{If } x_1 \text{ is } A_{j1} \text{ and } ... \text{ and } x_n \text{ is } A_{jn} \text{ then Class } C_j \text{ with } CF_j, \quad (1)$$

where R_j is the label of the *j*-th fuzzy rule, $\mathbf{x} = (x_1,...,x_n)$ is an *n*-dimensional pattern vector, A_{ji} is a fuzzy set for the *i*-th attribute, C_j is a consequent class, and CF_j is a certainty grade in the unit interval [0,1]. Since the consequent class C_j and the certainty grade CF_j of each fuzzy rule in (1) can be easily determined by a heuristic rule generation procedure from the given training patterns (see Ishibuchi 1992 for details of the rule generation procedure), the design of fuzzy rule-based systems is to determine the number of fuzzy rules and the antecedent part of each rule.

The main difference between two approaches employed in this paper lies in the coding method of fuzzy rules. We show characteristic features in the coding method of fuzzy rules and the rule generation method of two approaches in the following subsections. From the characteristic feature of the coding scheme, we refer to the first approach as rule selection method, and the second approach as fuzzy partition method.

2.1 RULE SELECTION METHOD

In the rule selection method, we use prespecified membership functions for fuzzy sets A_{ji} in (1), each of those are related to a linguistic term. Fig. 1 shows examples of such membership functions. Linguistic terms are denoted as S, MS, M, ML and L, that are related to membership functions. These linguistic values are used in our computer simulations for all attributes. Since this is just for simplicity of explanation, our first approach is applicable to more general cases where a different set of linguistic values is given to each attribute. In such a general case, membership functions are not necessary to be triangular. They are specified according to domain knowledge and intuition of human experts.

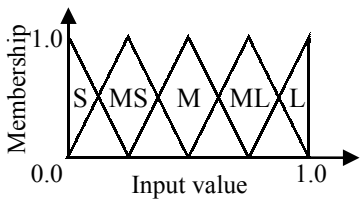


Figure 1. Membership functions of five linguistic values (S: *small*, MS: *medium small*, M: *medium*, ML: *medium large*, and L: *large*).

In the rule selection method (Ishibuchi 1997), all combinations of antecedent fuzzy sets were examined to generate candidate rules from which a small number of fuzzy rules were selected by genetic algorithms. This method cannot be directly applied to high-dimensional problems because the number of candidate rules exponentially increases with the dimensionality of pattern spaces. In order to reduce the number of fuzzy rules, we use a prescreening procedure of candidate rules. Our trick for prescreening candidate rules is based on the length (i.e., the number of antecedent conditions) of fuzzy rules.

While the total number of possible combinations of antecedent fuzzy sets is huge, the number of short fuzzy rules with only a few antecedent conditions is not large. Thus we can generate a tractable number of candidate rules by examining only short fuzzy rules. When we have five fuzzy sets, the number of fuzzy rules of the length k is calculated as ${}_nC_k \times 5^k$ which is the total number of combinations of selecting k attributes (i.e., ${}_nC_k$) and assigning linguistic values to the k selected attributes (i.e., 5^k). After generating a tractable number of fuzzy rules, we employ a binary string, where each bit corresponds to each fuzzy rule. The numeral "1" in each bit shows that the corresponding fuzzy rule is selected for a fuzzy rule-based classification system. On the other hand, the numeral "0" shows the corresponding fuzzy rule is not selected for a system. The aim of genetic algorithms is to design a fuzzy rule-based classification system by selecting appropriate fuzzy rules.

2.2 FUZZY PARTITION METHOD

In the fuzzy partition method, membership functions for fuzzy sets in (1) are directly coded as binary strings. In this method, the number and the shape of membership functions of each fuzzy set are simultaneously determined from training patterns.

Fig.2 shows an example of the fuzzy partition proposed in (Murata 1999). Membership functions of antecedent fuzzy sets are represented by a binary string. Membership functions of antecedent fuzzy sets on the i-th axis are denoted by a string $L_i = l_1 l_2 \cdots l_{k_i}$ with the length k_i. In Fig. 2, k_i is specified as $k_i = 10$. The value "1" in a string L_i indicates the existence of a membership function with the membership value 1.0 at the corresponding position (see Fig. 2). Neighboring membership functions always overlap each other at a membership value of 0.5. It should be noted that the longer the length k_i of the string L_i, the finer the tuning of each membership function.

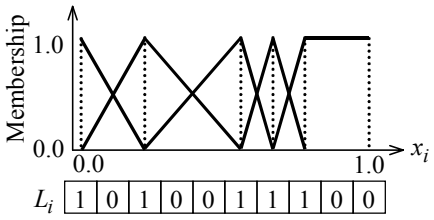


Figure 2. Membership functions in the fuzzy partition method.

The number of fuzzy if-then rules exponentially increases as the number of attributes increases. To remedy this difficulty, Murata *et al.* (1999) introduced a binary string I of the length n (the number of attributes) for selecting attributes by genetic algorithms. In the string I, each bit corresponds to each attribute. The i-th bit with the value "1" indicates the i-th attribute is selected for generating fuzzy if-then rules.

In order to construct fuzzy classification systems with multiple fuzzy rule tables, Murata *et al.* (1999) introduced a binary string T of the length t (the maximum number of fuzzy rule tables). In the string T, each bit corresponds to each table. The h-th bit with the value "1" indicates the h-th table is employed for the classification system. A binary string I in the above paragraph is multiplied so that each table has a different combination of input attributes. The string I_h indicates the selected attributes for the h-th table.

Since the strings I_h's and the string T for rule table selection are introduced for input selection, we handle the concatenated string $S = T \cdot I_1 I_2 \cdots I_t \cdot L_1 L_2 \cdots L_n$ as an individual in our genetic-algorithm-based fuzzy partition method. The concatenated string S specifies the fuzzy partition L_i ($i = 1, 2, ..., n$) of each of the selected fuzzy rule tables by T where selected attributes are denoted by I_h, $h = 1, 2, ..., t$. Since fuzzy rule tables are generated from the selected attributes and the specified fuzzy partition by S, S can be viewed as a rule set or a fuzzy rule-based classification system.

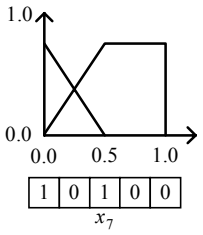
3 THREE-OBJECTIVE GA

As we have already mentioned, the comprehensibility of fuzzy rule-based system is impaired by the increase in the number of fuzzy rules. Thus we try to minimize the number of fuzzy rules. From the viewpoint of the comprehensibility of each fuzzy rule, a large number of antecedent conditions are not desirable. Thus we also try to minimize the number of antecedent conditions. At the same time, we want to design fuzzy rule-based systems with high classification performance. Based on these considerations, we formulate our task of designing comprehensible fuzzy rule-based classification systems as the following three-objective optimization problem:

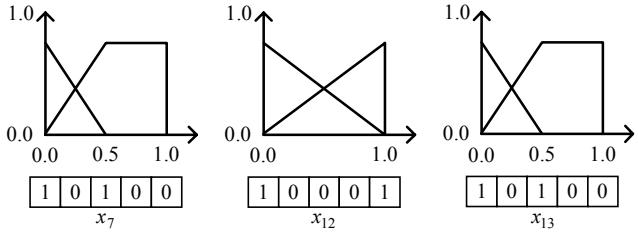
$$\text{Maximize } f_1(S), \text{ minimize } f_2(S) \ \& \text{ minimize } f_3(S), \quad (2)$$

where $f_1(S)$ is the number of correctly classified training patterns by a rule set S, and $f_2(S)$ is the number of fuzzy rules in S. In the rule selection method, $f_3(S)$ is the total number of antecedent conditions in S. For example, if we have a system with two fuzzy rules, where one rule has one antecedent condition and the other has three antecedent conditions, the total number of antecedent conditions in S is four. On the other hand, $f_3(S)$ is the total number of attributes used in constructed rule tables for the fuzzy partition method. If we obtain two rule tables shown in Fig. 3, the total number of attributes selected for the rule tables is four. In this case two fuzzy rules are obtained from Rule Table 1 (see Fig. 3 (a)), and eight fuzzy rules are generated from three attributes selected for Rule Table 2 (see Fig. 3 (b)), so we have ten fuzzy rules in total for the classification system designed by the fuzzy partition method. In order to obtain a compact fuzzy rule-based classification system, we should reduce the number of fuzzy rules. We can eliminate fuzzy rules from a classification system that classify no training patterns in a

classification problem. That is, we can reduce those fuzzy rules, because those have no effect on classification. After this rule reduction process, we can obtain classification systems with a small number of fuzzy rules.



(a) Rule Table 1 with a single attribute.



(b) Rule Table 2 with three attributes.

Figure 3. Two rule tables (the fuzzy partition method).

Let us briefly explain the concept of non-dominated rule sets for our three-objective optimization problem. A rule set S is said to be dominated by another rule set S^* if all the following inequalities hold:

$$f_1(S) \le f_1(S^*),\ f_2(S) \ge f_2(S^*),\ \&\ f_3(S) \ge f_3(S^*), \quad (3)$$

and at least one of the following three inequalities holds:

$$f_1(S) < f_1(S^*),\ f_2(S) > f_2(S^*),\ \text{or}\ f_3(S) > f_3(S^*). \quad (4)$$

The first condition (i.e., all the three inequalities in (3)) means that no objective of S^* is worse than S. The second condition (i.e., one of the three inequalities in (4)) means that at least one objective of S^* is better than S. If there exists no S^* that satisfies both the above two conditions, the rule set S is said to be a non-dominated rule set.

We employ three-objective genetic algorithms for finding non-dominated rule sets. Standard single-objective genetic algorithms are also applicable to our problem if the three objectives are integrated into a single scalar fitness function. Before describing three-objective genetic algorithms, we briefly discuss the handling of our problem by single-objective genetic algorithms.

A well-known simple trick for handling multi-objective optimization problems is to combine multiple objectives into a single scalar fitness function using weight parameters as

$$fitness(S) = w_1 \cdot f_1(S) - w_2 \cdot f_2(S) - w_3 \cdot f_3(S), \quad (5)$$

where w_1, w_2 and w_3 are non-negative real numbers. In (5), the two objectives $f_2(S)$ and $f_3(S)$ to be minimized can be viewed as having negative weights "$-w_2$" and "$-w_3$", respectively. The three weights w_1, w_2 and w_3 in (5) should be specified based on the users' preference in a particular pattern classification problem. It is, however, difficult to assign appropriate values to the three weights.

In this paper, we use a multi-objective genetic algorithm for finding non-dominate rule sets of our three-objective optimization problem. Multi-objective genetic algorithms do not require the specification of the weight parameters or the desired goals. We use a multi-objective genetic algorithm (Murata 1995) which is based on the scalar fitness function in (5) with random weight values. The weight values w_1, w_2, and w_3 are randomly updated whenever a pair of parent strings are selected. This is one characteristic feature of our multi-objective genetic algorithm. Another characteristic feature is that non-dominated rule sets are stored in a tentative pool separately from the current population. The tentative pool is updated at every generation in order to store only non-dominated rule sets among examined ones. From the tentative pool, N_{elite} rule sets are randomly selected as elite individuals, which are added to a new population. The outline of our three-objective genetic algorithm is written as follows.

[Three-objective genetic algorithm]

Step 1) Initialization: Generate an initial population of N_{set} rule sets where N_{set} is the population size.

Step 2) Evaluation: Calculate the values of the three objectives for each rule set in the current population. Then update the tentative pool of non-dominated rule sets.

Step 3) Selection: Repeat the following procedures to select ($N_{\text{set}} - N_{\text{elite}}$) pairs of rule sets.

a) Randomly specify the three weight values as

$$w_i = random_i / (random_1 + random_2 + random_3), \quad i = 1,2,3, \quad (6)$$

where $random_i$ is a non-negative random real number.

b) Calculate the fitness value for each solution by (5) using the randomly specified weight values. Then select a pair of rule sets based on the fitness value of each rule set. We specify the selection probability of each rule set S in the current population Ψ using the roulette wheel selection with the linear scaling:

$$P(S) = \frac{fitness(S) - f_{\min}(\Psi)}{\sum_{S \in \Psi} \{fitness(S) - f_{\min}(\Psi)\}}, \quad (7)$$

where $f_{\min}(\Psi)$ is the minimum value of the fitness values in the current population Ψ.

Step 4) Crossover and mutation: Generate a new rule set

from each pair of selected rule sets by crossover and mutation operations. These two operations are used with prespecified probabilities. By the genetic operations, ($N_{\text{set}} - N_{\text{elite}}$) rule sets are generated.

Step 5) Elitist strategy: Randomly select N_{elite} non-dominated rule sets from their tentative pool, and add them to the generated ($N_{\text{set}} - N_{\text{elite}}$) rule sets for constructing a new population of the size N_{set}.

Step 6) Termination test: If a prespecified stopping condition is satisfied, end the algorithm. Otherwise, return to Step 2.

The choice of crossover and mutation operations in Step 4 depends on the coding of rule sets. They are described in the following sections.

We use this three-objective genetic algorithm because it can be easily implemented. This algorithm involves no additional parameters. It uses only standard parameters such as population size, crossover probability, mutation probability, and the number of elite solutions. Other multi-objective genetic algorithms are also applicable to our three-objective optimization problems. For reviews of multi-objective genetic algorithms, see Veldhuizen 2000, Zitzler 1999, 2000.

4 RULE SELECTION METHOD

Let N be the number of generated candidate rules. A subset S of the N candidate rules is denoted by a binary string of the length N as $S = s_1 s_2 \cdots s_N$. In this coding, $s_j = 1$ and $s_j = 0$ mean that the j-th candidate rule R_j is included in S and excluded from S, respectively. The size of the search space with this coding is 2^N, which is the total number of subsets of the N candidate rules. Each rule set S is evaluated by the fitness function in (5) using randomly specified three weights whenever a pair of parent strings is selected. Since each rule set is represented by a binary string, standard genetic operations are applicable. In our computer simulations, we used the uniform crossover and the bit-change mutation.

For efficiently searching for small rule sets with high classification ability, we use two domain-specific techniques. One technique is a kind of local search. When the fitness value of a binary string S (i.e., rule set S) is calculated, all the given training patterns are classified by S for calculating the first objective $f_1(S)$. From the classification phase by the constructed classification system in Ishibuchi *et al.* (1992), a single winner rule is responsible for the classification of each training pattern. If a fuzzy rule in S is responsible for the classification of no training pattern, we can remove that rule without causing any deterioration of the first objective because that rule has no influence on the classification of any training pattern. At the same time, the elimination of such a fuzzy rule improves the second objective $f_2(S)$ and the third objective $f_3(S)$. Thus we remove all the fuzzy rules that are not responsible for the classification of any training pattern. This local search technique is applied to every rule set before its three objectives are evaluated in Step 2 of our three-objective genetic algorithm.

The other technique is to bias the mutation. A larger probability was assigned to the mutation from $s_j = 1$ to $s_j = 0$ than the mutation from $s_j = 0$ to $s_j = 1$. That is, the mutation is biased toward the decrease of the number of fuzzy rules in order to improve the second and third objectives. The biased mutation plays an important role especially when the number of candidate rules is large.

These two techniques are added to the three-objective genetic algorithm in the previous section.

5 FUZZY PARTITION METHOD

Since the fuzzy partition method has a concatenated strings $S = T \cdot I_1 I_2 \cdots I_t \cdot L_1 L_2 \cdots L_n$ as an individual solution, the specially designed genetic operations are employed for this method in Murata *et al.* (1999). The characteristic feature of the genetic algorithm with this coding method is the following:

1. *Crossover operation*: Selected attributes are interchanged between the parent strings, and each substring L_i is interchanged as a block.
2. *Mutation operation for tuning membership functions*: For the fine tuning of membership functions, the mutation operation that interchanges neighboring bits in each substring L_i is introduced. The adjustment of membership functions can be performed as in a local search procedure by slightly modifying their shapes and positions.
3. *Mutation operation for reducing the number of membership functions*: For decreasing the number of membership functions (i.e., the number of fuzzy if-then rules), the mutation probabilities for two directions (*i.e.*, $1 \to 0$ and $0 \to 1$) are not the same. The mutation probabilities for this mutation are specified as follows:
$$P_{\text{reverse}}(1 \to 0) > P_{\text{reverse}}(0 \to 1).$$
4. *Mutation operation for reducing the number of attributes*: For decreasing the number of selected attributes, the mutation probabilities for two directions (i.e., $1 \to 0$ and $0 \to 1$) are not the same. The mutation probabilities for this mutation are specified as follows:
$$P_{\text{input reverse}}(1 \to 0) > P_{\text{input reverse}}(0 \to 1).$$

6 COMPUTER SIMULATIONS

6.1 DATA SETS AND PARAMETERS

We applied the rule selection method and the fuzzy partition method to commonly used data sets in the literature: iris data, wine data, and glass data. All the data sets are available from the UC Irvine machine learning database.

Since we had no domain knowledge on each data set, we used the five linguistic values with the triangular

membership functions in Fig. 1 for every attribute of each data set in the rule selection method. The number and the shape of membership functions are automatically determined in the fuzzy partition method. Our two algorithms (i.e., the rule selection method and the fuzzy partition method) were executed under the framework of the three-objective genetic algorithm in Section 3 using the following parameter values. The population size was 20 rule sets, the number of elite solutions was six, and each algorithm was terminated at the 500th generation.

For the rule selection method in Section 4, we used the crossover probability: 0.9 and the biased mutation probabilities: 0.001 for the mutation from $s_j = 0$ to $s_j = 1$ and 0.1 for the mutation from $s_j = 1$ to $s_j = 0$. Randomly generated 20 initial rule sets were evolved in the rule selection method. For the fuzzy partition method, we employed the resolution $k_i = 5$ for each attribute, and we allowed ten rule tables at most in the fuzzy partition method. We used the following parameters: the crossover probability: 1.0, the mutation probability for tuning membership functions: 0.1, the biased mutation probabilities for reducing the number of membership functions: $P_{\text{reverse}}(1 \to 0) = 0.1$ and $P_{\text{reverse}}(0 \to 1) = 0.02$, and the biased mutation probabilities for reducing the number of attributes: $P_{\text{input reverse}}(1 \to 0) = 0.1$ and $P_{\text{input reverse}}(0 \to 1) = 0.05$. Our three-objective genetic algorithm searched for non-dominated rule sets by the evolution from 20 initial rule sets.

Each algorithm was applied to each data set 10 times. A set of non-dominated solutions stored in their tentative pool was obtained as final solutions of each trial with respect to our three objectives: the number of correctly classified training patterns, the number of fuzzy rules, and the total length of fuzzy rules in the rule selection method, or the total number of attributes used in selected rule tables in the fuzzy partition method. From 10 trials, we obtained 10 sets of non-dominated solutions. For concisely summarizing simulation results, we merged them into a single solution set and compared solutions with each other. In such comparison, some solutions were dominated by other solutions obtained from different trials. All solutions that were dominated by other solutions from different trials were removed from the enlarged solution set. The refined solution set is reported as simulation results by each algorithm for each data set in this section. Since the classification performance is measured by the number of correctly classified training patterns in our three-objective optimization problem, we report the value of this objective of each non-dominated solution together with the other two objective values. All the available training data were used in our computer simulations and the classification performance on those training data is reported in this section. Ishibuchi *et al.* (1999) reported the simulation results on the tradeoff between generalization ability of fuzzy rule-based systems and the number of fuzzy rules, where the classification performance on test data was evaluated by the leaving-one-out (LV-1) procedure and the ten-fold cross-validation (10-CV) procedure in computer simulations.

6.2 SIMULATION RESULTS ON IRIS DATA

The iris data set is a three-class pattern classification problem with four attributes and 150 patterns. We use the iris data set for illustrating three-objective rule selection while it is not actually a high-dimensional pattern classification problem. Fuzzy rules of the following type are used for the iris data set with four attributes:

$$\text{Rule } R_j: \text{ If } x_1 \text{ is } A_{j1} \text{ and } \ldots \text{ and } x_4 \text{ is } A_{j4} \text{ then Class } C_j \text{ with } CF_j. \quad (8)$$

Since the iris data set includes only four attributes, we can examine all the $(5+1)^4 = 1296$ combinations of antecedent linguistic values for generating candidate fuzzy rules for the rule selection method. By examining those combinations, we generated 587 fuzzy rules from the given 150 training patterns. Some fuzzy rules could not be generated because no training patterns were compatible with those rules. All the generated 587 fuzzy rules were used as candidate rules. In our rule selection method, each rule set was represented by a binary string of the length 587. For obtaining non-dominated rule sets of the three-objective optimization problem, we applied the three-objective rule selection method to the 587 candidate rules 10 times using different initial populations. From the 10 trials, we found seven non-dominated rule sets in Table 1. Since the iris data set is a three-class pattern classification problem, at least three fuzzy rules are necessary for designing fuzzy rule-based systems with high classification ability. In this sense, Table 1 includes three rule sets that are not practically useful. In Table 1, we can observe a tradeoff between the classification performance and the size of rule sets. We also applied the fuzzy partition method to the iris data set. Table 2 shows the obtained non-dominated solutions. Since the shape of the membership functions are adjusted in the fuzzy partition method, better rule sets are found with respect to the number of fuzzy rules and the number of correctly classified patterns. For example, the following classification system with three fuzzy rules was found by the rule selection method:

R_1: If x_3 is S then Class 1 with $CF_1 = 1.00$,

R_2: If x_3 is M then Class 2 with $CF_2 = 0.79$,

R_3: If x_4 is ML then Class 3 with $CF_3 = 0.70$.

By the above three fuzzy rules, 142 training patterns are classified correctly. From Table 2, we can see that a classification system with three fuzzy rules was also found by the fuzzy partition method. Fig. 4 shows the membership functions of the classification system. Since only a single attribute x_4 was selected for the rule table, membership functions can be seen as fuzzy rules. In this case, the total length of the rule set is three. Therefore the

fuzzy partition method found a better rule set with three rules than the rule selection method for the iris data set.

Table 1: Non-dominated rule sets obtained by the rule selection method for iris data.

Number of Rules	0	1	2	3	4	4	5
Total length	0	1	2	3	4	5	8
Number of Patterns	0	50	100	142	145	146	147
Rate (%)	0	33.3	66.7	94.7	96.7	97.3	98.0

Table 2: Non-dominated rule sets obtained by the fuzzy partition method for iris data.

Number of Rules	0	1	2	3	4	5
Number of Attributes	0	1	1	1	2	3
Number of Patterns	0	50	100	144	146	147
Rate (%)	0	33.3	66.7	96.0	97.3	98.0

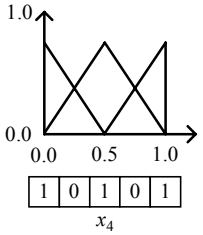


Figure 4. Rule table with three rules (fuzzy partition method).

6.3 SIMULATION RESULTS ON WINE DATA

The wine data set is a three-class pattern classification problem with 13 attributes and 178 patterns. It is impractical to generate candidate rules by examining all the $(5+1)^{13}$ combinations of antecedent linguistic values (i.e., about 13 billion combinations) in the rule selection method. Candidate rules were generated by examining only short fuzzy rules of the length 2 or less. Using this prescreening procedure, we generated 1834 candidate rules. The three-objective rule selection method was applied to the 1834 candidate rules 10 times using different initial populations. We obtained 21 non-dominated rule sets from the 10 trials. In Table 3, we show 7 non-dominated rule sets with high classification rates. Simulation results in Table 3 show that compact rule sets with a small number of short fuzzy rules were found by the rule selection method. Table 4 shows seven non-dominated solutions with high classification obtained by the fuzzy partition method. Fig. 3 shows the obtained classification system with six rules and 4 attributes. This system classifies 174 training patterns correctly.

6.4 SIMULATION RESULTS ON GLASS DATA

In the previous computer simulations, we have already shown that the rule selection method can find a small number of short fuzzy rules with high classification performance for designing comprehensible fuzzy rule-based systems. In this subsection, we examine the rule selection method and the fuzzy partition method through computer simulations on glass data. The glass data set is a six-class pattern classification problem with nine attributes and 214 patterns. The glass data set is a difficult classification problem with large overlaps between different classes in the pattern space. So it may be difficult to design compact fuzzy rule-based systems with high classification performance by a small number of short fuzzy rules.

From 10 trials, we obtained 22 non-dominated rule sets. Table 5 shows seven non-dominated rule sets with high classification rates. From this table, we can see that fuzzy rule-based systems with high classification rates could not found. We also applied the fuzzy partition method to the glass data. We obtained 27 non-dominated solutions and show seven rule sets with high classification rates in Table 6. We can also see that classification systems with high classification rates could not found. We employed finer resolution $k_i = 11$ in the fuzzy partition method. Table 7 shows seven rule sets with high classification rates. We can see that the fuzzy partition method with finer resolution could find better classification systems with respect to the classification performance.

Table 3: Non-dominated rule sets obtained by the rule selection method for wine data.

Number of Rules	6	5	7	6	7	8	10
Total length	7	9	9	10	12	14	18
Number of Patterns	173	173	174	175	176	177	178
Rate (%)	97.2	97.2	97.8	98.3	98.9	99.4	100

Table 4: Non-dominated rule sets obtained by the fuzzy partition method for wine data.

Number of Rules	4	4	6	4	7	5	6
Number of Attributes	2	3	3	4	3	4	4
Number of Patterns	166	169	170	171	172	172	174
Rate (%)	93.3	94.9	95.5	96.1	96.6	96.6	97.8

Table 5: Non-dominated rule sets obtained by the rule selection method for glass data.

Number of Rules	8	9	10	10	11	12	14
Total length	12	14	16	17	18	20	25
Number of Patterns	150	151	152	153	154	155	156
Rate (%)	70.1	70.6	71.0	71.5	72.0	72.4	72.9

Table 6: Non-dominated rule sets obtained by the fuzzy partition method for glass data.

Number of Rules	6	9	19	10	24	12	13
Number of Attributes	5	3	3	4	3	4	4
Number of Patterns	144	146	147	148	149	152	153
Rate (%)	67.3	68.2	68.7	69.2	69.6	71.0	71.5

Table 7: Non-dominated rule sets obtained by the fuzzy partition method for glass data (fine resolution).

Number of Rules	20	22	27	29	31	32	69
Number of Attributes	6	3	3	3	3	3	4
Number of Patterns	162	164	165	166	167	169	171
Rate (%)	75.7	76.6	77.1	77.6	78.0	79.0	79.9

7 CONCLUSION

In this paper, we described and compared two genetic-algorithm-based approaches for finding non-dominated solutions (i.e., non-dominated fuzzy rule-based systems) with respect to the three objectives. We can see that the rule selection method could find compact classification systems with high performance for the wine data set. Since the search space of the fuzzy partition method is larger than that of the rule selection method, the classification systems with the large number of rules are found (for example, 28 rules in Table 6). We may find more compact classification systems by the fuzzy partition method if we have enough computation time for the method.

References

C.M.Fonseca and P.J.Fleming (1995). An overview of evolutionary algorithms in multiobjective optimization, *Evolutionary Computation* 3: 1-16.

D.E.Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

J.Horn, N.Nafpliotis and D.E.Goldberg (1994). A niched Pareto genetic algorithm for multi-objective optimization. *Proc. of 1st IEEE International Conference on Evolutionary Computation*: 82-87.

H.Ishibuchi, K.Nozaki and H.Tanaka (1992). Distributed representation of fuzzy rules and its application to pattern classification. *Fuzzy Sets and Systems* 52: 21-32.

H.Ishibuchi, T.Murata, I.B.Turksen (1997). Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems* 89: 135-149.

H.Ishibuchi, T.Sotani and T.Murata (1999). Tradeoff between the performance of fuzzy rule-based classification systems and the number of fuzzy if-then rules. *Proc. of 18th Internaitonal Conference of the North American Fuzzy Information Processing Society – NAFIPS '99*: 125-129.

Y.Jin (2000). Fuzzy Modeling of High-Dimensional Systems: Complexity Reduction and Interpretability Improvement. *IEEE Transactions on Fuzzy Systems* 8: 212-221.

Y.Jin, W.von Seelen, B.Sendhoff (1999). On Generating FC^3 Fuzzy Rule Systems from Data using Evolution Strategies. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics* 29: 829-845.

F.Kursawe (1991). A variant of evolution strategies for vector optimization. In H.-P.Schwefel and R.Männer (Eds.), *Parallel Problem Solving from Nature*. 193-197. Berlin: Springer-Verlag.

T.Murata and H.Ishibuchi (1995). MOGA: Multi-objective genetic algorithms. *Proc. of 2nd IEEE International Conference on Evolutionary Computing*: 289-294.

T.Murata, H.Ishibuchi and M.Gen (1999). Construction of fuzzy classification systems using multiple fuzzy rule tables. *Proc. of 1999 IEEE International Conference on System, Man and Cybernetics*: CD-ROM.

V.de Oliveira (1999). Semantic Constraints for Membership Function Optimization. *IEEE Trans. on Systems, Man, and Cybernetics - Part A: Systems and Humans* 29: 128-138.

W.Pedrycz, V.de Oliveira (1996). Optimization of Fuzzy Models. *IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics* 26: 627-637.

J.D.Schaffer (1985). Multi-objective optimization with vector evaluated genetic algorithms. *Proc. of 1st International Conference on Genetic Algorithms*: 93-100.

M.Setnes, R.Babuska, U.Kaymak, H.R.van Nauta Lemke (1998a). Similarity Measures in Fuzzy Rule Base Simplification. *IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics* 28: 376-366.

M.Setnes, R.Babuska, B.Verbruggen (1998b). Rule-Based Modeling: Precision and Transparency. *IEEE Trans. on Systems, Man, and Cybernetics - Part C: Applications and Reviews* 28: 165-169.

M.Setnes, H.Roubos (2000). GA-Fuzzy Modeling and Classification: Complexity and Performance. *IEEE Trans. on Fuzzy Systems* 8: 509-522.

D.A.van Veldhuizen, G.B. Lamont (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation* 8: 125-147.

J.Yen, L.Wang (1999). Simplifying Fuzzy Rule-Based Models using Orthogonal Transformation Methods. *IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics* 29: 13-24.

J.Yen, L.Wang, W.Gillespie (1998). Improving the Interpretability of TSK Fuzzy Models by Combining Global Learning and Local Learning. *IEEE Trans. on Fuzzy Systems* 6: 530-537.

E.Zitzler, K.Deb and L.Thiele (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation* 8: 173-195.

E.Zitzler and L.Thiele (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto Approach. *IEEE Trans. on Evolutionary Computation* 3: 257-271.

Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for Pareto selection

Jason Noble
Informatics Research Institute
School of Computing, University of Leeds
Leeds LS2 9JT, UK
jasonn@comp.leeds.ac.uk

Richard A. Watson
DEMO Lab, Computer Science Dept.
Brandeis University
Waltham, MA 02454, USA
richardw@cs.brandeis.edu

Abstract

When using an automatic discovery method to find a good strategy in a game, we hope to find one that performs well against a wide variety of opponents. An appealing notion in the use of evolutionary algorithms to coevolve strategies is that the population represents a set of different strategies against which a player must do well. Implicit here is the idea that different players represent different "dimensions" of the domain, and being a robust player means being good in many (preferably all) dimensions of the game. Pareto coevolution makes this idea of "players as dimensions" explicit. By explicitly treating each player as a dimension, or objective, we may then use established multi-objective optimization techniques to find robust strategies. In this paper, we apply Pareto coevolution to Texas Hold'em poker, a complex real-world game of imperfect information. The performance of our Pareto coevolution algorithm is compared with that of a conventional genetic algorithm and shown to be promising.

1 INTRODUCTION

One of the inherent problems with learning game strategies through self-play is a tendency for such strategies to be brittle—to be over-specialised to a particular area of strategy space—and to fail to find robust, general strategies (see, e.g., Pollack & Blair, 1998, for discussion). The potential for strategies to have intransitive superiority relationships is an important key for understanding why this might happen. That is, although some player A might be beaten by some other player B, and B may in turn be beaten by C, it may not be the case that C beats A (Cliff & Miller, 1995). The existence of such intransitive superiority relationships can mean that although a search method persistently finds strategies that are better than the last strategy, it fails to find a strategy that is good in general. Intransitive superiority relationships suggest that a problem domain is multi-dimensional, in the sense that being good against one strategy does not necessarily mean that you are good against another (Watson and Pollack, this volume).

An appealing notion in the use of evolutionary algorithms to coevolve strategies is that the population represents a set of different strategies against which a player must do well. Implicit here is the idea that different players represent different "dimensions" of the domain, and being a robust player means being good in many (preferably all) dimensions of the game. However, the idea that players represent dimensions of the game remains implicit in standard coevolutionary algorithms. Pareto coevolution makes the concept of "players as dimensions" explicit. By explicitly treating each player as a dimension, or objective, we may then apply established multi-objective optimization techniques—in particular, principles such as Pareto dominance—to find robust strategies. This may help to prevent the effects of intransitive superiority from interfering with the discovery of good general solutions, because multi-objective optimization promotes a set of players with a different balance of abilities rather than promoting the single best-on-average strategy. Pareto coevolution was explored by Watson and Pollack (2000), and follows from work relating coevolution and Pareto dominance (Ficici & Pollack, 2000). Pareto coevolution is also developed in the domain of the cellular automata majority problem by Ficici and Pollack (2001). In this paper, we apply Pareto coevolution to Texas Hold'em poker.

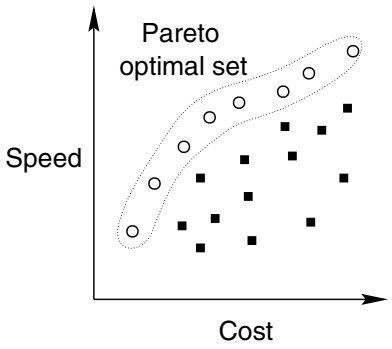


Figure 1: Solution points for a hypothetical car design problem, in which we want to maximize speed and minimize cost. The Pareto-optimal set is indicated.

1.1 PARETO SELECTION AND GAMES

A Pareto-optimal solution is one in which none of the relevant measurements or dimensions of quality or performance can be improved without reducing performance on one or more of the other dimensions. For example, if we were designing a car, and our goals were low cost and a high top speed, there might be a Pareto-optimal solution at $20,000 and 120 mph. This means that 120 mph is the fastest you can go for that price, and that $20,000 is the cheapest you can pay for that speed. An alternative design with the same price but a top speed of only 110 mph would clearly be inferior. However, there will almost always be more than one solution in the Pareto-optimal set of best possible compromises (see Figure 1). Perhaps there are also Pareto-optimal design possibilities at $25,000 and 130 mph, and at $15,000 and 100 mph. The spirit of the Pareto approach is not to somehow convert dimensions like speed and cost into a common currency in order to come up with the one true optimum, but to find all members of the Pareto-optimal set so that a human decision-maker, or some other method, can be allowed to choose between them.

Within the field of evolutionary computation, various methods of approximating the Pareto-optimal set have been proposed as tools for multi-objective optimization (for reviews see, e.g., Fonseca & Fleming, 1995; Horn, 1997). The details differ, but, in essence, Pareto dominance is used as a selection criterion. Candidate solution A Pareto-dominates solution B if A is at least as good as B on all dimensions, and better than B on one or more. Pareto selection involves choosing the *non-dominated* solutions for reproduction.

Pareto selection is typically carried out with respect to a small number of dimensions, as in the car example above. This paper seeks to apply Pareto selection to the domain of games (von Neumann & Morgenstern, 1953) by using each player in an evolving population as a dimension, or objective, to be optimized—hence, Pareto coevolution.

Given a particular game, and a way of representing strategies in that game, we could list every possible strategy. We could also observe the performance of each strategy against every other, and the matrix so derived would allow us to see that some strategies Pareto-dominate others, e.g., that A performs as well as B when playing C, D, and E, and is better than B when playing F. We could then spell out the membership of the Pareto-optimal set. It is important to realize that the set might include surprising members: perhaps a strategy that does very poorly on average would nevertheless be included because of its exceptional performance against just one opponent.

The brute-force approach of calculating a performance matrix of all-against-all will work for a sufficiently simple game with a small number of possible strategies, but it obviously will not be feasible for games of any complexity. The size of the performance matrix will be equal to the number of possible strategies squared, and reliably calculating each entry in the matrix will require many trials if the game includes a stochastic element.

We have utilized a population-based coevolutionary approach, in which individual strategies from a population of modest size are selected at random to compete against each other for a number of trials. The accumulated data from many of these trials can be seen as a noisy, partial window onto the true performance matrix. Non-dominated strategies are preserved in a Pareto front, and novel strategies are generated through sexual reproduction of strategies in the front. In this way we hoped that our population would come to approximate the true Pareto-optimal set, and would provide robust general strategies.

1.2 HOLD'EM POKER AS A TEST CASE

To provide a convincing test of the hypothesis that Pareto coevolution can be used to find robust strategies, we wanted to avoid toy problems in favour of a real game. We have chosen poker, a card game of some depth in which a wide range of strategies and skill levels are exhibited by human players.

The specific poker variant we used was limit Texas Hold'em, one of the most popular versions of poker in modern casinos. The popularity of Hold'em must be partly due to the balance between public and private

information in the game, which leaves a lot of room for convincing bluffs. A game of Texas Hold'em typically involves eight to ten players, and each complete hand has the following four-round structure.

The pre-flop: each player is dealt two cards face down. These are hole cards, or private cards. The player to the dealer's left makes a forced bet called the small blind, equal to one chip in our case. The next player must bet the big blind, which is equal to two chips. The third and subsequent players must then call (match the bet), raise (increase the bet) or fold (throw in their cards and forfeit all interest in the pot). As this is a limit game, any raises must be exactly two chips at this stage. In addition, no more than three raises are allowed in this or any other round of betting, unless there are only two players left, in which case raising can continue until someone runs out of chips.

The flop: when the previous round of betting is complete (all players have either called or folded), three cards are "flopped" face up in the middle of the table. These are community cards, and are available to all players. By mentally combining the community cards with their hole cards, players can now form a 5-card poker hand, such as two pair, or a flush. There is another round of betting, again starting with the player to the dealer's left. Players can check (decline to bet if no-one else has bet), call, raise by two chips only, or fold.

The turn: a fourth community card is turned face up, and there is another round of betting. Note that even though six cards are now available, players can only make five-card poker hands. The stakes increase now, and all raises must be four chips.

The river: a fifth community card is dealt face up, and there is a final round of betting, with four-chip raises. When the round of betting is complete, all players who still have an interest in the pot compare their hands, and the player with the strongest hand[1] takes the pot.

The art of the game consists of such points as knowing when your cards are likely to be strongest, knowing whether it's worth staying in the pot to improve your hand with subsequent community cards, reading the likely strength of your opponents' hands through their patterns of betting, and of course effective bluffing (see Sklansky, 1999, for a more authoritative discussion).

[1]Poker hands, from weakest to strongest, are: high card, a pair, two pair, three of a kind, a straight, a flush, a full house, four of a kind, and a straight flush.

2 METHODS

2.1 REPRESENTING POKER STRATEGIES

Our primary goal was to test the effectiveness of our Pareto coevolution algorithm, not to evolve world-class poker strategies. We have therefore used an economical representation scheme that is not able to capture many of the subtleties of expert-level poker. In deciding whether to fold, call, or raise, our strategies attend to the strength of their hand at each point in the game. They do not pay any attention to the behaviour of other players except insofar as they are aware of what the current bet is, and may choose to fold because the stakes have become too high for them.

The strategy representation begins with two probability values (real numbers between zero and one inclusive). The first gives the probability with which a player will bluff (i.e., pretend to have very strong cards) on any given hand. The second gives the probability with which a player will check-raise when given the opportunity—this is a deceptive play in which a player bets nothing, indicating weakness, and then raises when the bet comes around again.

Next there are 24 integers in groups of six, describing strategy for each of the four betting rounds (see section 1.2). Two integers describe the minimum cards that a player wants at this stage in order to remain in the hand, e.g., a pair of aces, three sevens, or a king-high flush. Another two integers describe the cards that a player would regard as a strong hand. One integer describes the amount that a player would prefer to bet at this stage, and a final integer gives the maximum amount a player will bet. If players have less than their minimum requirements, they will check if possible or fold if asked to bet. If players have equalled or exceeded their minimum requirements, they will raise until the betting reaches their preferred level. If betting goes higher than their preferred level, they will call until their maximum bet is exceeded, and then they will fold. But if their cards qualify as strong, they will call any bet.

Finally, four groups of four binary values modify the player's behaviour on each betting round. One bit indicates whether or not the player will ignore their normal preferred and maximum bets, and instead bet as much as they possibly can, if their cards qualify as strong. A second bit determines whether or not the

player is willing to stay in the hand if their cards are no better than what is showing on the community cards (for example, if the player holds ace-king, and the flop is three queens, then the player's hand is three queens, but that hand is available to all the other players too). A third and a fourth bit indicate a willingness to stay in the hand if one card short of a straight or a flush respectively. (Note that the second bit does not apply to the pre-flop round, and the third and fourth bits do not apply to the pre-flop round or the river round.)

Some of the features that a more sophisticated strategy representation might cover include: whether or not the two pre-flop cards are the same suit (for possible flushes) or close in value (for possible straights), the player's position in the betting order, whether a player has paired the top, middle or bottom pair on the flop, the relative size of the player's stack of chips, whether the size of the pot justifies a risky bet, and how often other players are seen to fold early or to bluff. Nevertheless, as is apparent to us from playing against various evolved and hand-coded strategies,[2] the current strategy representation is adequate to produce poker strategies ranging from the very bad to the reasonably good.

2.2 A SIMPLE PARETO COEVOLUTION ALGORITHM

We began with a population of 100 random poker strategies. Ten strategies were selected at random to make up a table, and a game of 50 hands of poker was played out. Two hundred such games were played per generation, which meant that each strategy was assessed over an average of 1000 hands, and had a chance to play against most of the other strategies in the population.

Results from each of the 10,000 hands of poker played in a generation were collated in a matrix showing who had won or lost chips to whom. Pairwise comparisons were conducted on this matrix in order to identify Pareto-dominated strategies. Non-dominated strategies were maintained in a Pareto front, and the remaining slots in the population were filled through sexual reproduction of randomly chosen members of the Pareto front. Reproduction included multi-point crossover and mutation as in a standard genetic algorithm (GA).[3] After the population had been restocked, the win-lose matrix was wiped clean, and the cycle began again.

One problem that became apparent in trial runs was that the entire population, or very close to it, would often be included in the Pareto front. This was presumably due in part to noise in our evaluation process—even over 1000 hands, the luck of the deal had a significant influence on success, making the true worth of a strategy hard to discern. Furthermore, each strategy could expect only about 100 hands against each opponent, and sometimes did not get to play against a specific opponent at all.

In order to keep exploring new regions in strategy space, we needed to limit the size of the Pareto front. We set the maximum size of the front at 50 strategies, which meant that up to half the population was preserving accumulated wisdom, while the other half was exploring new possibilities. But in the event that more than 50 strategies were non-dominated at the end of a generation's 10,000 hands, we needed a principled way of deciding which strategies would be maintained in the front and which would be discarded. In devising a metric for this purpose, we wanted to stay as close as possible to the Pareto selection ideal, i.e., that one should not assume that the dimensions of success are equally weighted. Strictly speaking, the method we devised does violate this—and we suspect that any method for keeping less than the full Pareto front must—but it does not use an average or sum of scores across different dimensions. Instead we have used a count on the number of dimensions in which a player excels.

Our method was to eliminate those strategies that were "nearly dominated," until our front size was less than or equal to 50. A strategy is nearly dominated if the number of opponents that it is superior to, with respect to its best competitor, is low. The best competitor is defined as the strategy that minimizes this number of opponents. To elaborate: in determining whether a strategy A is Pareto-dominated by B, or vice versa, we look at the scores of A and B against all other strategies. We count the number of strategies, or dimensions, for which A scores higher than B. If this count is zero, then A is dominated by B, and will not be a member of the Pareto front in any event—there is

[2] Code (in C) for playing poker against evolved and hand-coded strategies is available on the web at http://www.comp.leeds.ac.uk/jasonn/Research/Pareto/ . Code for running our Pareto selection algorithm is also available.

[3] There were 37 genetic loci, the crossover rate was 0.1 per locus, and the mutation rate was 0.02 per locus. For the genetic parameters that were real or integer values, mutation was implemented as a small gaussian perturbation, with a mean of zero and a standard deviation of 0.05, 1, or 2 for probabilities, hand rankings, and betting amounts respectively (see section 2.1 for details). Ten percent of mutations were denoted as catastrophic and resulted in a new random value for that parameter.

nothing that A can do that B cannot do better. If this count is greater than zero, then A is not dominated by B. If we look at these counts for A compared with all other strategies, the minimum count gives an indication of how close A came to being dominated. In order to limit the size of the front, we throw out strategies for which this count was equal to one, then two, then three, etc., until the membership of the Pareto front is less than or equal to 50.

In summary, our implementation of Pareto coevolution involved selection based on non-dominance, given the noisy, partial window onto the true payoff matrix that is obtained from the results of a generation's 10,000 poker hands. We also developed a heuristic for limiting the size of the Pareto front. However, there were potential problems with our procedure. Although a strategy that is dominated with respect to the current population must also be dominated with respect to all possible strategies, the converse is not true (Schaffer, 1985). So one strategy might remain in our Pareto front despite being dominated by another, as yet unseen (or already discarded). Noise in the evaluation process, combined with our elimination heuristic, might prevent non-dominated strategies from being recognized as such in the first place. Another possible complication is perhaps specific to the game of poker: success is measured in the context of the other players at a table, but this is not explicitly controlled for. Strategy A might tend to do very well against strategy B when matched directly, but not at a table where C and D were present.

2.3 MEASURING EFFECTIVENESS OF THE ALGORITHM

In order to determine the effectiveness of our Pareto coevolution procedure, we compared its performance with that of a regular coevolutionary GA. This merely provides a baseline performance measure to give us an indication of whether Pareto coevolution can improve performance and robustness of evolved strategies, compared to regular coevolution where fitness is based on an average score over opponents in the population.

The parameters for the GA (i.e., population size, number of hands played per generation, mutation rate, crossover rate, etc.) were the same as those used for the Pareto coevolution algorithm. Strategies were selected for reproduction based on their profit or loss after 10,000 hands: specifically, the scores were normalized with the minimum set equal to zero, and roulette-wheel selection applied to the normalized scores.

Both algorithms were run 20 times for 100 generations each time. We can view the comparison of the two algorithms as a test of which one can produce the best strategies given a million hands (100 generations $\times$ 10,000 hands) worth of information.

In deciding which of the two algorithms had produced better results, we were faced with a somewhat paradoxical problem of measurement. Precisely because the fitness of a strategy cannot be given in isolation, but can only be measured with respect to a particular opponent or set of opponents, it is difficult for us to provide a single, general measure of the strength of the evolved strategies. The familiar Red Queen effect means that it will not help to look at performance against the other strategies in the population, as the zero-sum nature of poker ensures that mean fitness will always be zero.

We decided to construct two sets of five hand-coded reference strategies for the purposes of comparison, using the same representational scheme as the evolving populations (see section 2.1). These reference strategies are not claimed to be in any way optimal; they merely represent some typical, more-or-less reasonable playing styles. For example, we constructed several conservative strategies, that would not bet unless they had quite strong cards. Some of the strategies were deceptive, either because of frequent bluffing, or through "slowplaying," i.e., hiding the strength of one's cards until late in the hand. Other strategies tended to call all bets as long as they held a reasonable hand.

Assessment of the strategies evolved under our two different selection regimes was carried out by having each strategy in the population play alone against a table stocked with reference strategies, for a fixed sequence of 1000 hands. The overall profit or loss of each evolved strategy was recorded. The same random seed was used to deal out the same sequence of cards in every assessment run, in an attempt to reduce some of the noise inherent in the process. The reference strategies were divided into an alpha and a beta group, and assessment was carried out against each of these groups. Note that the ten reference strategies were simply sorted at random into the two assessment groups; there was no intention that the alpha group should be superior to the beta group, for instance. We wanted to be sure that we had not accidentally constructed an unusual or eccentric reference point, and comparison of results against two distinct groups gave us some some insurance against this possibility.

It is important to be clear about what good performance against these two reference groups might mean. Strategies under both selection regimes never encountered any of the reference strategies during the course of evolution. Strategies were selected solely for their

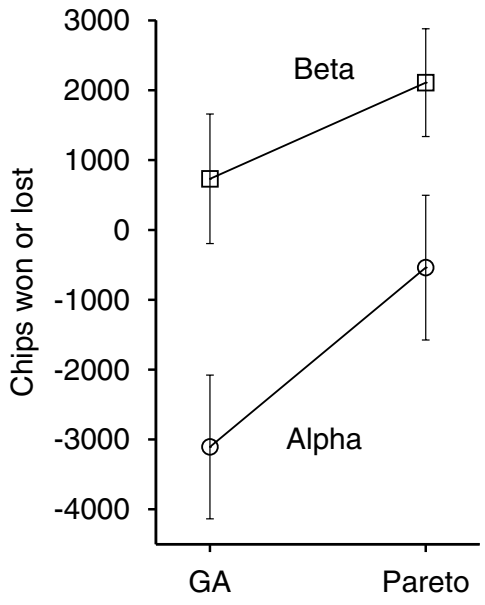


Figure 2: Mean performance (±1 standard error) of evolved strategies in 1000 hands of play against the alpha and beta reference groups; strategies evolved with a coevolutionary GA compared with strategies evolved under Pareto coevolution. Results summarised across 20 runs in each case.

ability to do well against other members of their population, either in the Pareto sense or in the conventional sense of having a high average score. If they managed to do well against an arbitrary set of hand-coded strategies, that gives us some indication that they would do well against a wide range of strategies, i.e., that they are robust and have not adapted to their conspecifics in an overly brittle manner.

3 RESULTS

Figure 2 shows that after 100 generations of evolution, strategies evolved under Pareto selection had a higher mean performance against both of the reference groups than did the strategies evolved using a conventional GA. As the standard error bars indicate, this difference is more pronounced in performance against the alpha group.

Figure 2 also indicates that the alpha reference group was significantly harder to beat than the beta group—both strategies lose to the former and win from the latter on average. This difference was not intended, but the fact that there is no evidence of a strong interaction between selection regime and reference group performance (i.e., the two lines in Figure 2 are roughly parallel) is a reassuring indicator that the two reference groups are measuring something like general ability.

If we look in detail at the evolved strategies across the two selection regimes, the most striking difference is that the Pareto strategies bluffed less often on average (20% vs. 36%). This fact alone explains a lot of the difference in success between the two conditions: in those populations where a high level of bluffing obtained, performance against the reference strategies was always very poor. This is because the only type of bluffing available to these strategies was a simple-minded approach in which they pretended they had a royal flush right from the beginning of the hand and never gave up their bluff no matter how determined the opposition. The Pareto selection process seems to have made it easier for the population to discover the folly of this sort of bluffing.

There were other differences: the Pareto strategies had lower standards for staying in at the preflop and at the river. They tended to bet more, and were more likely to bet as much as possible if they had strong cards (except on the final round of betting). They were less likely to stay in the hand if they weren't beating the community cards, and were more likely to wait for straights and flushes if they were one card short. Readers who play poker may be interested in seeing a complete strategy description. The following is a high-performing evolved strategy from Pareto run 17, in which the average wins were 2009 and 5267 chips against the alpha and beta groups respectively.

- Never bluff, and check-raise 11% of the time.
- At the pre-flop stage, bet as much as possible if you have an ace or a pair—otherwise fold.
- On the flop, stay in as long as you are beating the community cards. If you are one short of a straight or a flush, stay in in any event. Try to bet just two chips, but call bets up to 42 chips. If you have a straight or better, call any bet.
- On the turn, keep waiting for a straight or a flush, but otherwise fold if you have less than a pair of sixes or if you are not beating the community cards. If you stay in, try to bet 6 but call bets up to 59 chips. If you have two pair, with the top pair sixes or better, bet as much as you can.
- On the river, if you have a pair of aces or better, then bet as much as possible. Otherwise fold, and definitely fold if your two aces are community cards.

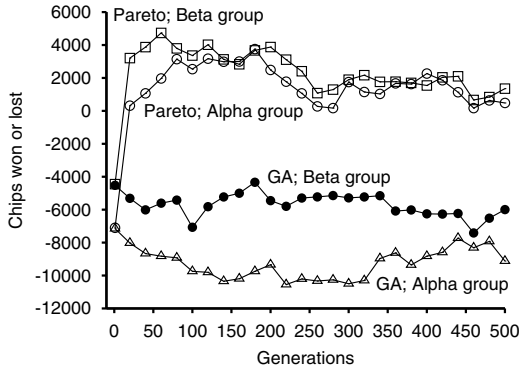


Figure 3: Mean performance of evolving strategies in 1000 hands of play against the alpha and beta reference groups, for both Pareto and standard coevolution, over 500 generations. Data taken from runs with a common random seed value of zero.

Some peculiarities were noted regarding the Pareto selection condition. The Pareto front, of maximum size 50, was always full, which means that some non-dominated strategies were being eliminated in every generation. The mean age of strategies in the Pareto front was approximately two generations, and the median age was always one generation. This suggests a front made up of mostly very young strategies with a few older ones, which is not unexpected, but the mean age of only two generations indicates an extremely rapid turnover of strategies. When Pareto populations were examined at the end of a run, they were not as diverse as we would have hoped. Again, this was not a complete surprise, as reproduction with crossover was employed, but it indicates that the Pareto front has not been completely successful in preserving a range of very different strategies that are non-dominated with respect to each other.

We looked briefly at what happened when evolution continued for more than 100 generations, and found that in many cases performance against the reference groups actually worsened. Figure 3 gives an example of this, with mean performance data over time for an extended version of run zero, showing both Pareto and standard coevolution against the two reference groups. The Pareto-evolved players are declining in performance and moving closer to zero profit, while the GA strategies are making significant losses but with no clear trend up or down.

4 CONCLUSIONS

Our Pareto coevolution algorithm was superior to a GA at finding robust Texas Hold'em strategies within 100 generations. This fact should not be over-interpreted: clearly, we worked with only one game, two small groups of arbitrary reference strategies, and a particular set of parameter values. Nevertheless, our finding does show that Pareto coevolution of strategies in games can work in principle and is an idea worth exploring.

The algorithms we have presented for selecting non-dominated strategies and for discarding excess strategies from the Pareto front could probably be improved upon so as to use the multi-dimensional information from the games played more efficiently and effectively. Our current method maintains only a rough approximation to the Pareto front as compared to existing multi-objective optimization methods, because of the unusually high number of objectives we are using. However, in regular coevolution the multidimensional information is discarded completely, in favor of a single "performance on average" dimension. To put it another way, fitness evaluation and selection are noisy, incomplete processes under both selection regimes—noisy because of the stochastic element, and incomplete in the sense that we cannot observe performance against all possible opponents. But in Pareto coevolution, we are trying to use the information gained from 10,000 hands of poker more intelligently: instead of simply taking an average, we use the specifics of who beat who, and we remove the unwarranted assumption that every other strategy is equally worth beating.

The long term behaviour shown in Figure 3 is somewhat disturbing. It seems that our Pareto-selected strategies cannot hold onto their collective wisdom over time (although the same effect was observed with the more successful GA-evolved strategies). This effect may be due to the population chasing its own tail into eccentric regions of the strategy space; if this is the case, then we need to refine our coevolution algorithm. But note that we are not selecting for maximization of scores against the reference strategies—we are selecting for not being dominated by anyone else in the population. It is an open question as to whether the long term reduction in success apparent in Figure 3 is a sign of "population senility." It may represent a movement towards careful compromise strategies that do not make spectacular wins, but instead make modest profits against a wide range of opponents, and are careful not to lose to anyone.

This paper is the preliminary exploration of an idea,

and so we have many questions for future work. One of the most pressing is about the explore-exploit balance in our algorithm: is 50% of the population a reasonable size for the ongoing Pareto front? Would we benefit from having a "genetic freezer" for storing past champion strategies, and then re-inserting them into the front at regular intervals? How big is the true Pareto-optimal set likely to be in a game like poker, and what chance do we have of getting a reasonable approximation to it with our method?

We also want to look at reproduction of Pareto-selected strategies. In the current paper we have used standard sexual reproduction, partly to facilitate comparison with the GA. It seems worth exploring asexual reproduction, or at least much lower levels of crossover, to see if we can avoid the unfortunate degree of convergence reported in section 3. It would be interesting to see whether asexual reproduction also resulted in an increase in the mean age of in the Pareto front.

Once we have refined our Pareto coevolution algorithm, it would be sensible to test it against more than just a standard GA. If we view the problem as how to learn the most you can from one million hands of poker, then we should ultimately be testing Pareto coevolution against a range of established evolutionary computation and machine learning techniques. In the meantime, our experiments have provided a simple illustration of Pareto coevolution, and begun to explore some of the issues involved.

Acknowledgments

The authors are grateful to Sevan Ficici for invaluable discussion and insight into the principles of Pareto selection as applied to games, and to Anthony Bucci for assistance in developing the heuristic to reduce the Pareto front. We would also like to thank four anonymous reviewers for their comments.

References

Cliff, D., & Miller, G. F. (1995). Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations. In Morán, F., Moreno, A., Merelo, J. J., & Chacón, P. (Eds.), *Advances in Artificial Life: Third European Conference on Artificial Life (ECAL '95)*, Vol. 929 of *Lecture Notes in Artificial Intelligence*, pp. 200–218. Springer, Berlin.

Ficici, S. G., & Pollack, J. B. (2000). A game-theoretic approach to the simple coevolutionary algorithm. In Schönauer, M., et al. (Eds.), *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN VI)*, pp. 467–476. Springer, Berlin.

Ficici, S. G., & Pollack, J. B. (2001). Pareto optimality in coevolutionary learning. Computer science technical report CS-01-216, Brandeis University.

Fonseca, C. M., & Fleming, P. J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, *3*(1), 1–16.

Horn, J. (1997). Multicriteria decision making and evolutionary computation. In *Handbook of Evolutionary Computation*. Institute of Physics Publishing, London.

Pollack, J. B., & Blair, A. D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, *32*(3), 225–240.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pp. 93–100. Lawrence Erlbaum.

Sklansky, D. (1999). *The Theory of Poker* (Fourth edition). Two Plus Two Publishing, Henderson, NV.

von Neumann, J., & Morgenstern, O. (1953). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton.

Watson, R. A., & Pollack, J. B. (2000). Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In Schönauer, M., et al. (Eds.), *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN VI)*, pp. 425–434. Springer, Berlin.

Escaping Hierarchical Traps with Competent Genetic Algorithms

Martin Pelikan
Depts. of General Engineering and Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
pelikan@illigal.ge.uiuc.edu

David E. Goldberg
Dept. of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801
deg@uiuc.edu

Abstract

To solve hierarchical problems, one must be able to learn the linkage, represent partial solutions efficiently, and assure effective niching. We propose the *hierarchical Bayesian optimization algorithm* which combines the Bayesian optimization algorithm, local structures in Bayesian networks, and a powerful niching technique. Additionally, we propose a class of hierarchically decomposable problems, called *hierarchical traps*, which are deceptive on each level. The proposed algorithm is shown to scale up subquadratically on all test problems. Empirical results are in agreement with recent theory.

1 INTRODUCTION

Genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989) combine short partial solutions to form solutions of higher order. New solutions undergo selection and the process is repeated until the entire solution is formed. However, fixed, problem-independent, recombination operators have shown to perform quite poorly on problems with interactions among the variables spread across the solutions (Thierens & Goldberg, 1993; Pelikan, Goldberg, & Cantú-Paz, 1998). Moreover, the hierarchical nature of the optimization process has earned only little attention and it has been assumed that genetic algorithms do this automatically.

The purpose of this paper is to show that competent genetic algorithms which succeeded in solving problems of bounded difficulty on a single level quickly, accurately, and reliably, can be extended to solve problems that are hierarchical in their nature. We focus on the Bayesian optimization algorithm (BOA) (Pelikan et al., 1998) using decision graphs to represent the conditional probabilities of the model used to represent promising solutions (Pelikan et al., 2000).

There are three major issues one must address to succeed in solving difficult hierarchical problems: linkage learning, niching, and efficient representation of the model. Linkage learning ensures powerful recombination. Niching and efficient representation of the model ensure preservation of alternative partial solutions that are assembled to form solutions of higher order. We propose the *hierarchical Bayesian optimization algorithm* which addresses the three aforementioned issues by combining the Bayesian optimization algorithm, local structures in Bayesian networks, and a powerful niching technique. Hierarchical BOA is able to solve problems that are not only hierarchical, but that also mislead the algorithm toward some inferior optimum on each level. Additionally, we design a class of hierarchical test problems which require both efficient linkage learning and niching, and perform a number of experiments to show that hierarchical BOA is able to solve the problems efficiently. Due to their deceptive nature, the proposed problems are called *hierarchical deceptive traps*.

The paper starts by describing BOA, which uses Bayesian networks to model promising solutions and generate the new ones. Section 3 discusses the use of niching in genetic and evolutionary algorithms. Hierarchical BOA is described in Section 4. Test problems tackled in our experiments are presented in Section 5. Section 6 provides and discusses the results of our experiments. Section 7 concludes the paper.

2 BAYESIAN OPTIMIZATION ALGORITHM

By applying recombination and mutation, GAs are manipulating a large number of promising partial solutions. However, fixed, problem independent, recombination and mutation operators often result in inferior performance even on simple problems. Without knowing where the important partial solutions are and

designing problem specific operators that take this information into account, the required number of fitness evaluations and population size grow exponentially with the number of decision variables (Thierens & Goldberg, 1993).

That is why there has been a growing interest in *linkage learning* which studies methods that are able to learn where the important interactions in the problem are and use this information to combine solutions more effectively. One of the approaches to linkage learning is based on using probability distributions to model promising solutions found so far and generating new solutions according to the estimated distribution (Mühlenbein & Paaß, 1996; Pelikan, Goldberg, & Lobo, 2000). Probability distributions can capture variables which are correlated and the ones which are independent. This can subsequently be used to combine the solutions in more effective manner. An overview of methods based on this principle is beyond the scope of this paper and can be found in Pelikan, Goldberg, and Lobo (2000) and other related papers.

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998) uses Bayesian networks to model promising solutions and subsequently guide the exploration of the search space. In BOA, the first population of strings is generated randomly with a uniform distribution. The initial population can be biased to the regions that we are interested in. From the current population, the better strings are selected. Any selection method can be used. A Bayesian network that fits the selected set of strings is constructed. Any metric as a measure of quality of networks and any search algorithm can be used to search over the networks in order to maximize/minimize the value of the used metric. Besides the set of good solutions, prior information about the problem can be used in order to enhance the estimation and subsequently improve convergence. New strings are generated according to the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones.

The next subsection describes basic principles of learning and utilization of Bayesian networks. Subsequently, local structures that can be used to make the representation of the model more efficient are discussed and a simple greedy algorithm for network construction is briefly described.

2.1 BAYESIAN NETWORKS

A Bayesian network (Pearl, 1988) is a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in our case, to the positions in solution strings). Mathematically, a Bayesian network encodes a joint probability distribution. A directed edge relates the variables so that in the encoded distribution, the variable corresponding to the terminal node is conditioned on the variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the corresponding variable with a conjunctional condition containing all its parents. The network encodes independence assumptions that each variable is independent of any of its antecedents in ancestral ordering given its parents.

To encode the conditional probabilities corresponding to the nodes of the network, one can use a simple probability table listing probabilities of all possible instances of a variable and its parents. However, the size of such a table grows exponentially with the number of parents of the variable even though many probabilities of higher order may be the same. To solve hierarchical problems, it is essential to be able to represent conditional probabilities by structures that are polynomial in the order of interactions. While the order of interactions can be as high as the size of the problem, the number of corresponding alternative partial solutions must be polynomial in their order to allow efficient and reliable exploration. The next subsection presents alternative ways to represent conditional probabilities in the model which allow a more compact representation of the local densities in the model.

2.2 LOCAL STRUCTURES IN BAYESIAN NETWORKS

One simple extension of the probability table is a *default table* (Friedman & Goldszmidt, 1999). In the default table, only some instances of the variable and its parents are listed together with the corresponding probabilities. The remaining probabilities are obtained from the default entry which is simply an average of the remaining (unlisted) probabilities.

One can use more complex local structures, such as *decision trees* (Friedman & Goldszmidt, 1999) or *decision graphs* (Chickering, Heckerman, & Meek, 1997). Each internal node of a decision tree or graph corresponds to some variable. Children (successors) of each internal node correspond to disjoint subsets of values the variable can obtain. For binary variables, each non-leaf node can have exactly two children where each child corresponds to one of the values zero and one. In case of bigger alphabets, there are more possibilities. A decision graph allows different parents to have the same child. This makes the structure both more general and expressive than decision trees. In hierarchical BOA we use decision graphs. However, there is only little difference between the performance

using decision graphs and trees. Since trees are simpler to interpret we used only decision trees in our experiments.

Using local structures can reduce the space we need to represent the model. Additionally, it can refine the model building by using smaller operators and make the model more general for some data sets. One can encode interactions of a high order without having to consider exponentially many instances and probabilities. Please, see Pelikan et al. (2000) for more details on using decision graphs in the BOA.

2.3 LEARNING BAYESIAN NETWORKS

To construct the network, a simple greedy algorithm is usually used. This algorithm performs simple graph operations that improve the quality of the current network the most, starting from an empty network or a network from a different source. To measure quality of each network, various scoring metrics can be used. Recently, we have used the Bayesian-Dirichlet metric, the minimum description length (MDL) metric, and a metric which is a combination of the Bayesian-Dirichlet and MDL metric. For more details on the network construction and scoring metrics for simple Bayesian networks and for networks with local structures, see Pelikan et al. (1998) and Pelikan et al. (2000). The next section discusses various approaches to niching. Subsequently, hierarchical BOA is described.

3 NICHING

The purpose of niching in genetic and evolutionary optimization is twofold: (1) discovery of multiple solutions of the problem and (2) preservation of alternative solutions until one can decide which solution is better. In some real-world applications it is important to find multiple solutions and let the expert or experiment decide which of the solutions is the best after all. The reason for preserving multiple alternative solutions is that on some difficult problems one cannot clearly determine which alternative solutions are really on the right track until the optimization proceeds for a number of generations. Without niching the population is a subject to genetic drift which may destroy some alternatives before we find out whether or not they are the ones we are looking for.

There are three general approaches to niching: fitness-sharing, selection-based, and island models. The following paragraphs briefly discuss each approach. It is beyond the scope of this paper to give a complete overview, and we refer the reader to the extended version of this paper (Pelikan & Goldberg, 2001).

The first approach modifies the fitness landscape before the selection is performed. Fitness sharing (Goldberg & Richardson, 1987) is based on this idea. In fitness sharing, the location of each individual is set to either its genotype or phenotype. The neighborhood of each individual is defined by the *sharing function*. An individual shares a niche with any individual that is within a certain range from its location. The effect may decrease with the distance and completely vanishes for distances greater than a certain threshold.

The second approach modifies the selection itself to take into account the fitness as well as the genotype or the phenotype instead of using the fitness as the only criterion. In *preselection* of Cavicchio (1970) the offspring replaced the inferior parent. This scheme was later generalized by De Jong (1975) who proposed *crowding*. In crowding, for each new individual a subset of the population is first selected. The new individual then replaces the most similar individual in this subset. Harik (1994) proposed the restricted tournament selection as an extension of De Jong's crowding. RTS proceeds just as crowding; however, the individual replaces the closest individual from the selected subset only if it is better in terms of fitness. Therefore, RTS introduces selection pressure and can replace the selection operator.

The third approach is to isolate several groups of individuals rather than to keep the entire population in one location. The location of each individual does not depend on its genotype or phenotype. The individuals can migrate between different locations (islands or demes) at certain intervals and allow population at each location to develop in isolation. There are two reasons why spatial separation should be desirable in genetic and evolutionary computation. One reason is that in nature the populations are actually divided in a number of subpopulations that (genetically) interact only rarely or do not interact at all. Another reason is that separating a number of subpopulations allows an effective parallel implementation and is therefore interesting from the point of view of computational efficiency.

Some related work studies the preservation of diversity from a different point of view. The primary goal of these techniques is not the preservation of multiple solutions or alternative search regions, but the avoidance of premature convergence. Various techniques for niching were also proposed in the area of multiobjective optimization. These methods are not applicable to single-criterion optimization and therefore we do not discuss them in this paper.

The BOA uses the set of selected solutions to learn a model of promising solutions. Fitness sharing would

affect the fitness and subsequently also the model construction. That is why it is desirable that we use a different niching method in the BOA. Spatial separation can be directly encoded in the probabilistic model by using mixture distributions or models with hidden variables. A simple method based on mixture models to reduce negative effects of symmetry in the problem on the BOA was proposed in Pelikan and Goldberg (2000a). However, to solve hierarchical problems, we must deal with a number of niches that can be exponential in the number of variables. Even though this implies exponentially sized populations, one can use the fact that the model itself preserves diversity quite well by that it makes many independence assumptions and uses these to generate new solutions. Only little extra pressure toward diversity preservation is then required. That is why we used the restricted tournament selection to incorporate niching into hierarchical BOA. Since the technique is used as a replacement technique and not as a primary source of selection pressure, we called the method *restricted tournament replacement.*

4 HIERARCHICAL BOA

As it was discussed above, hierarchical BOA uses Bayesian networks to learn the linkage. To efficiently represent partial solutions, local structures are used to represent local densities in the model. The remainder of this section describes restricted tournament replacement (RTR) used in hierarchical BOA to ensure effective niching.

RTR localizes the replacement in hierarchical BOA by selecting a sub-set of the original population for each new offspring and letting the offspring compete with the most similar member of this subset. If the new offspring is better, it replaces the corresponding individual. The measure of similarity can be based on either the genotype or the phenotype. In our experiments, we used Hamming distance to measure similarity. Since the generation of a probabilistic model in the BOA does not encourage using a steady state genetic algorithm, we incorporate niching in the replacement step of a traditional BOA.

It is important to set the size of the subsets that are selected to incorporate each new individual into the original population. The size of these subsets is called a *window size*. A window size should be proportional to the number of niches. We have tried a number of settings on various difficult problems. A window size proportional to the size of the problem yielded the best performance.

A window size proportional to the size of the problem can be supported by the following argument. For correct decision making on a single level, the population size must grow proportionally to the problem size (Pelikan, Goldberg, & Cantú-Paz, 2000). To maintain a certain number of niches, one must lower-bound the size of each niche by a certain constant. Therefore, a population size proportional to the problem size allows for maintenance of the number of niches proportional to the problem size. The number of niches that RTR can maintain is proportional to the window size. Therefore, the window size growing linearly with the size of the problem is the strongest niching one can afford without increasing population sizing requirements.

5 TEST PROBLEMS

In order to analyze the performance of hierarchical BOA on difficult hierarchical problems, most test problems are hierarchical. The remainder of this section describes test problems used in our experiments.

5.1 HIERARCHICALLY DECOMPOSABLE FUNCTIONS

Hierarchically decomposable functions (HDFs) (Watson, Hornby, & Pollack, 1998; Pelikan & Goldberg, 2000b) are a subclass of general additively decomposable functions (Pelikan, Goldberg, & Cantú-Paz, 1998). HDFs are defined on multiple levels where the input to each level is based on the solutions found on lower levels. The *fitness contribution* of each building block is separated from its *interpretation* (meaning) when it is used as a building block for constructing the solutions on a higher level. The overall fitness is computed as the sum of fitness contributions of each building block.

In spite of bounded difficulty of HDFs on each level, a hierarchical function can contain interactions of order equal to the size of the problem. Bounded difficulty on each level of the hierarchy makes HDFs solvable in polynomial time even though the problem is very difficult when viewed on a single level. It is important to note that hierarchical problems of bounded difficulty are a strictly more difficult class of problems than problems of bounded difficulty on a single level.

A hierarchically decomposable function is defined by its structure in the form of a tree with one-to-one mapping between the leaves and the variables in a problem, and two sets of functions: (1) the interpretation functions and (2) the contribution functions. The structure defines which blocks of interpretations to interpret to the next level and how, and which blocks contribute to the overall fitness on this level. The interpretation functions define how we interpret solutions from lower

levels to become inputs of the contribution and interpretation functions on a higher level. The contribution functions define how much do blocks of interpretations on each level contribute to the overall fitness.

The difficulty of hierarchical functions depends on the underlying structure as well as the contribution and interpretation functions. The hierarchical if-and-only-if (HIFF) function (Watson et al., 1998) uses the "if and only if" function on each level. More difficult functions have been proposed (Goldberg, 1997; Goldberg, 1998; Pelikan & Goldberg, 2000b), where functions deceive the algorithms to a local optimum on each level. Only at the top level it becomes clear which optimum is the global one.

In this paper, only simple structures such as balanced binary and ternary trees are used. The contribution of each subfunction on each level is scaled so that the contributions on all levels are of the same magnitude.

5.1.1 Hierarchical If-and-Only-If (HIFF)

The structure of the HIFF is a balanced binary tree. By $height(x)$ we denote the distance from the node x in the tree to one of its descendant leaves. Since the tree is balanced, the height is well-defined. Each leaf contributes to the fitness by 1. Each parent node x contributes to the overall fitness by $2^{height(x)}$ if and only if the interpretations of its children are both either 0 or 1. Otherwise, the contribution is 0. The two symbols are interpreted to their parent on the next level as 0 in case they are both 0's, 1 in case they are both 1's, and '-' otherwise. As input, the leaves of the tree get the input string with no change.

5.1.2 Hierarchical Trap Functions

Hierarchical traps use a balanced k-ary tree as the underlying structure, where $k \geq 3$. The interpretation functions interpret blocks of all 0's and 1's to 0 and 1, respectively, similarly to the HIFF. Everything else is interpreted into '-'.

Each contribution function is a trap function of order k. A trap function is a function of unitation, i.e. its value depends only on the number of ones in the input string. See Figure 1 for a graph of the trap function of order k. If there is a '-' in the input to this function, it simply returns 0.

The values of f_{high} and f_{low} define the heights of the two peaks. The trap function is fully deceptive whenever f_{high} is greater than f_{low} within some proportion depending on the order k of the function. See Deb and Goldberg (1994) for sufficient conditions of deception. If the function is deceptive or $f_{low} > f_{high}$, any schemata of order lower than k bias the search to the string all zeroes.

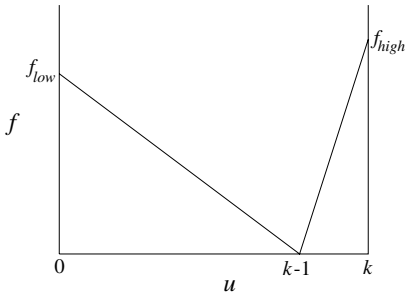


Figure 1: Trap function of order k.

In both functions used in our experiments, the underlying structure is a ternary tree ($k = 3$) and the leaves do not directly contribute to the overall fitness. For all non-leaf nodes x of the first hierarchical trap except for the root, the contribution is computed by a trap with equal peaks $f_{high} = f_{low} = 1$ multiplied by $3^{height(x)}$. The contribution of the root node is given by a trap with $f_{high} = 1$ and $f_{low} = 0.9$ multiplied by $3^{height(root)}$. In this fashion, the function biases the search to the solution of all zeroes on each but the top level. However, the optimum is in the string of all ones. The top level is also deceptive which makes the problem even harder. The above function is denoted by H-Trap1 in further text.

In the second function the bias toward solutions with many zeroes is made even stronger by making the peak f_{low} higher than the other peak everywhere except for the root. To keep the global optimum in the string of all ones, we set $f_{high} = 1$ and $f_{low} = 1 + 0.1/k$ for all non-root levels. This function is denoted by H-Trap2.

The HIFF function does not bias the search toward either global optimum. Unlike the HIFF, both hierarchical trap functions H-Trap1 and H-Trap2 bias the search toward the solution with all zeroes on all levels. However, the actual global optimum is in the string of all ones. Therefore, the functions are very difficult to solve and without effective linkage learning required to preserve the local optima on each level and niching required to preserve alternative partial solutions until solving the problem on the highest level, the algorithm cannot reach the global optimum. For a more detailed description of the test functions, see Pelikan and Goldberg (2001).

5.2 BIPOLAR FUNCTION

The bipolar deceptive function of order 6 is constructed by concatenating a number of bipolar subfunctions of order 6 (Deb, Horn, & Goldberg, 1992). See Pelikan and Goldberg (2001) for a full definition of the function. The bipolar function of size n has $2^{n/6}$

global and $20^{n/6}$ local optima. For $n = 30$, there are 32 global and $3,200,000$ local optima.

6 RESULTS

To show how hierarchical BOA scales up on difficult hierarchical problems, we performed tests on each function with varying problem size. For each problem size, we required that the algorithm find the global optimum in all 30 independent runs. The performance was measured by an average number of fitness evaluations until the optimum was found. The population size was determined empirically to minimize the number of fitness evaluations until the optimum was found. A window size was set to the problem size, i.e. $w = n$. We used decision trees to represent conditional probabilities in the model and construct the model. Prior distribution of models was biased toward simpler models (Pelikan et al., 2000).

The results of our experiments on the HIFF and H-Trap1 functions are shown in Figure 2. In all three cases the algorithm scales up subquadratically. On the left-hand side of the figure, the graphs in arithmetic scale display the growth of the number of fitness evaluations with respect to the size of the problem for the HIFF and H-Trap1 problems. Results on H-Trap2 were within 8% of the results on H-Trap1 and due to the lack of space we do not present them here (see Pelikan and Goldberg (2001)).

Theory of population sizing and time to convergence for the BOA on separable problems of bounded difficulty (Pelikan et al., 2000) can be used to estimate time to convergence of hierarchical BOA on hierarchical problems. Theory suggests that a single level of the hierarchical problem can be solved in about $O(n^{1.5})$ fitness evaluations. The number of levels in all three hierarchical problems grows as $O(\log n)$. Thus, the overall time to convergence should grow as $O(n^{1.5} \log n)$. The fit is very good and matches also the slopes in the log-log scaled graphs very accurately.

On the right-hand side, the log-log scaled graphs including the slopes between neighboring points are shown. A linear function in this scale is a polynomial of the degree equal to the slope of the curve. To show that the number of fitness evaluations grows at most polynomially with the problem size, the points must lie on a straight line. In our experiments, we see that the slopes in fact *decrease* with the problem size. This is the effect of the logarithm in the expected number of fitness evaluations.

The simple genetic algorithm with fixed crossover is not able to optimize hierarchical functions without making sure that interacting genes are close to each other. Under the assumption of tight linkage, the simple genetic algorithm with good niching should work quite well. The algorithm presented in Watson (2000) is able to solve the HIFF problem even for interacting genes spread throughout the strings. However, Watson's algorithm requires $O(n^2 \log n)$ fitness evaluations for a problem of size n which is more than is required by our algorithm.

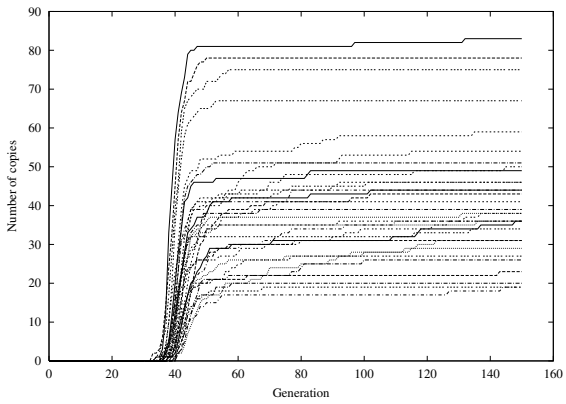
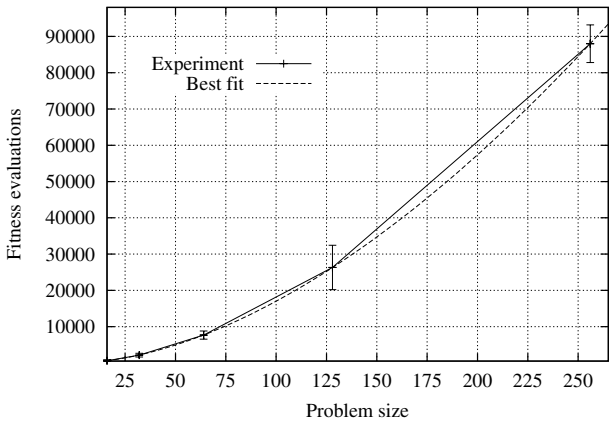


Figure 3: Number of copies of different global optima of the bipolar function. There are 32 optima in this function and all 32 are multiply represented at the end of the run.

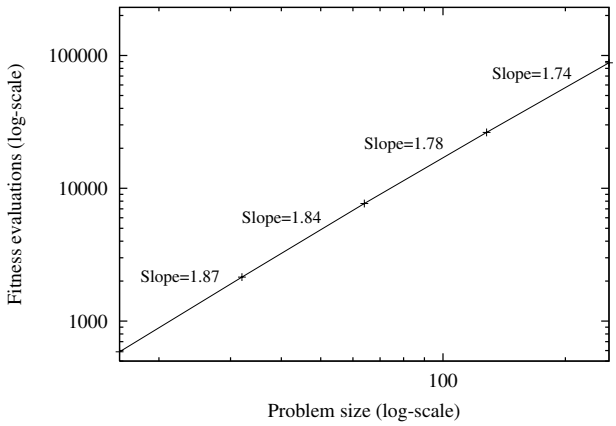
To show the ability of hierarchical BOA to discover multiple optima, we also performed a single run on a bipolar function of size $n = 30$ with a sufficiently big population and recorded the number of copies of each global optimum in the population (see Figure 3). We have performed a number of experiments with varying parameters with a very similar result. The algorithm was able to discover and maintain all global optima which soon took over the entire population. However, the optima were not equally distributed, ranging from about 1.27% to about 5.53% of the population. This confirmed the intuition that, unlike fitness sharing, the methods based on crowding are not very sensitive to the fitness values. They are able to maintain a number of alternatives but the total space occupied by each alternative is not proportional to its fitness.

7 CONCLUSIONS

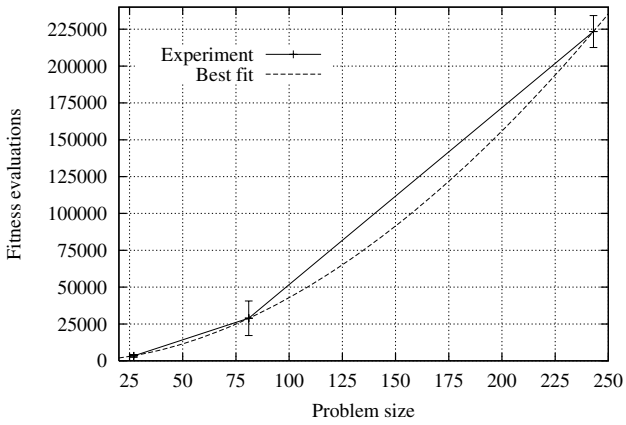
The paper takes another important step toward increasingly competent genetic algorithms by providing an algorithm that is able to solve problems on a single level as well as multiple levels. It emphasizes the importance of solving separable problems on a single level by showing that we need not modify much to successfully move from a single level to hierarchies. To solve



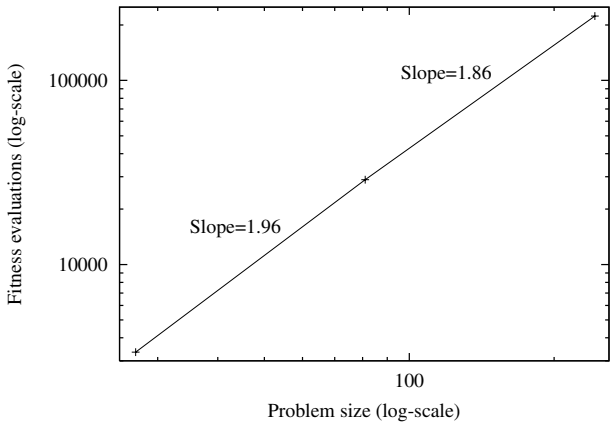
(a) HIFF: Arithmetic scale.



(b) HIFF: Log scale.



(c) H-Trap1: Arithmetic scale.



(d) H-Trap1: Log scale.

Figure 2: Results on the hierarchical functions.

hierarchically decomposable problems quickly, accurately, and reliably, a combination of niching, linkage learning, and efficient representation of partial solutions is necessary.

To learn the linkage, hierarchical BOA uses Bayesian networks to model promising solutions and to generate the new ones. To efficiently represent partial solutions, decision graphs are used to represent local densities in a model. To assure powerful niching, the restricted tournament replacement is used.

Separable deceptive problems of bounded difficulty are extended to multiple levels. The designed hierarchical trap problems that are deceptive on each level are intractable by local search methods and can be used as a benchmark for other optimization algorithms. Hierarchical BOA can solve these problems very efficiently and reliably and it scales up subquadratically with the problem size. Population sizing and convergence theory can be used to approximate the behavior of the algorithm both on single-level and hierarchical problems.

Hierarchical BOA should be applicable to real-world problems without problem specific knowledge ahead of time. This takes us closer to the promised land of robustness, that has long been associated with GAs but rarely delivered.

Acknowledgments

The authors would like to thank Martin Butz, Erick Cantú-Paz, Clarissa Van Hoyweghen, Fernando Lobo, Franz Rothlauf, and Kumara Sastry for many useful discussions and valuable comments.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-94-1-0103, F49620-95-1-0338, F49620-97-1-0050, and F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The US Government

is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Martin Pelikan was partially supported by grant VEGA 1/7654/20 of Slovak Grant Agency. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U.S. Government.

References

Cavicchio, Jr., D. J. (1970). *Adaptive search using simulated evolution*. Doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 25-0199).

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

Deb, K., & Goldberg, D. E. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, *10*, 385–408.

Deb, K., Horn, J., & Goldberg, D. E. (1992). *Multimodal deceptive functions* (IlliGAL Report No. 92003). Urbana, IL: University of Illinois at Urbana-Champaign.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (1 ed.). (pp. 421–459). Cambridge, MA: MIT Press.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (1997, November). *The design of innovation: Lessons from genetic algorithms*. Unpublished manuscript.

Goldberg, D. E. (1998, June 15). Four keys to understanding building-block difficulty. Presented in Projet FRACTALES Seminar at I.N.R.I.A. Rocquencourt, Le Chesnay, Cedex.

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 41–49). Hillsdale, NJ: Lawrence Erlbaum Associates.

Harik, G. (1994, May). *Finding multiple solutions in problems of bounded difficulty* (IlliGAL Report No. 94002). Urbana, IL: University of Illinois at Urbana-Champaign.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Eiben, A., et al. (Eds.), *Parallel Problem Solving from Nature - PPSN IV* (pp. 178–187). Berlin: Springer Verlag.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, California: Morgan Kaufmann.

Pelikan, M., & Goldberg, D. E. (2000a). *Genetic algorithms, clustering, and the breaking of symmetry* (IlliGAL Report No. 2000013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., & Goldberg, D. E. (2000b). *Hierarchical problem solving by the Bayesian optimization algorithm* (IlliGAL Report No. 2000002). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., & Goldberg, D. E. (2001). *Escaping hierarchical traps with competent genetic algorithms* (IlliGAL Report No. 2001003). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1998). *Linkage problem, distribution estimation, and Bayesian networks* (IlliGAL Report No. 98013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000). *Bayesian optimization algorithm, population sizing, and time to convergence* (IlliGAL Report No. 2000001). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*. In press.

Pelikan, M., Goldberg, D. E., & Sastry, K. (2000). *Bayesian optimization algorithm, decision graphs, and Occam's razor* (IlliGAL Report No. 2000020). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.

Watson, R. A. (2000). Analysis of recombinative algorithms on a non-separable building-block problem. *Foundations of Genetic Algorithms*. In printing.

Watson, R. A., Hornby, G. S., & Pollack, J. B. (1998). Modeling building-block interdependency. In Eiben, A. E., et al. (Eds.), *Parallel Problem Solving from Nature, PPSN V* (pp. 97–106). Berlin: Springer Verlag.

Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor

Martin Pelikan
Dept. of General Engineering
and Dept. of Computer Science
University of Illinois, Urbana, IL 61801
pelikan@illigal.ge.uiuc.edu

David E. Goldberg
Dept. of General Engineering
University of Illinois
Urbana, IL 61801
deg@uiuc.edu

Kumara Sastry
Dept. of General Engineering
University of Illinois
Urbana, IL 61801
kumara@illigal.ge.uiuc.edu

Abstract

This paper discusses the use of various scoring metrics in the Bayesian optimization algorithm (BOA) which uses Bayesian networks to model promising solutions and generate the new ones. The use of decision graphs in Bayesian networks to improve the performance of the BOA is proposed. To favor simple models, a complexity measure is incorporated into the Bayesian-Dirichlet metric for Bayesian networks with decision graphs. The presented modifications are compared on a number of interesting problems.

1 INTRODUCTION

Recently, the use of local structures, such as default tables and decision trees/graphs, in context of learning the structure of Bayesian networks has been proposed and discussed (Friedman & Goldszmidt, 1999; Chickering, Heckerman, & Meek, 1997). Using local structures has shown to improve the performance of learning in terms of the likelihood of the resulting models on a number of benchmark data sets. However, none of these approaches was used to improve model building in the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1998), which uses Bayesian networks to model promising solutions and guide the search. Moreover, the use of various metrics in the BOA has not been investigated thoroughly.

The purpose of the paper is twofold. First, the use of decision graphs in the model construction phase of the BOA is proposed to improve its performance. Second, to eliminate superfluously complex models, a model complexity measure is incorporated into the Bayesian-Dirichlet scoring metric for Bayesian networks with decision graphs. It is empirically shown that the introduced pressure is sufficient to eliminate the necessity of binding the complexity of models by the user. This is a significant contribution. The advantages of the minimum description length (MDL) metric, which favors simple models, are attained without having to sacrifice a possibility of using prior knowledge about the solved problem introduced by Bayesian metrics. The performances of the BOA with various scoring metrics and network construction algorithms, and the simple genetic algorithm are compared on a number of problems.

The paper starts by describing the BOA. Section 3 provides basic theoretical background of learning a structure of Bayesian networks. Section 4 describes how decision graphs can be used as a core component of learning the structure of Bayesian networks and provides a Bayesian scoring metric for computing the marginal likelihood of Bayesian networks using decision graphs given the data. The results of our experiments are described in Section 5. The paper is concluded in Section 6.

2 BAYESIAN OPTIMIZATION ALGORITHM

It has been shown that using recombination and selection is a very powerful approach for optimizing many difficult problems. However, fixed, problem independent, recombination and mutation operators often result in inferior performance even on simple problems. Without knowing where the important interactions in the problem are and designing problem specific operators that take this information into account, the number of fitness evaluations and the required population sizes grow exponentially with the number of decision variables (Thierens & Goldberg, 1993).

Much effort was put in the design of methods that would be able to learn which parts of the solutions should be combined and which ones should remain

intact. One of the approaches replaces traditional crossover and mutation by building a probabilistic model of promising solutions and using this model to generate offspring. Probability distributions can capture variables which are correlated and the ones which are independent. This can subsequently be used to combine the solutions in more effective manner. Methods based on this principle are called estimation of distribution algorithms (Mühlenbein & Paaß, 1996), probabilistic model-building genetic algorithms (PMBGAs) (Pelikan, Goldberg, & Lobo, 2000), or iterated density estimation algorithms (Bosman & Thierens, 2000).

It is beyond the scope of this paper to give a complete overview of PMBGAs and the interested reader should refer to Pelikan et al. (2000). In this paper we focus on the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998) which uses Bayesian networks to model promising solutions and subsequently guide the exploration of the search space.

In the BOA, the first population of strings is generated randomly with a uniform distribution. From the current population, the better strings are selected. Any selection method can be used. A Bayesian network that fits the selected set of strings is constructed. Any metric as a measure of quality of networks and any search algorithm can be used to search over the networks in order to maximize/minimize the value of the used metric. Besides the set of good solutions, prior information about the problem can be used in order to enhance the estimation and subsequently improve convergence. New strings are generated according to the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones.

3 BAYESIAN NETWORKS

A Bayesian network (Pearl, 1988) is a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in our case, to the positions in solution strings). Mathematically, a Bayesian network encodes a joint probability distribution. A directed edge relates the variables so that in the encoded distribution, a variable corresponding to the terminal node is conditioned on a variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the corresponding variable with conjunctional condition containing all its parents. The network encodes independence assumptions that each variable is independent of any of its antecedents given its parents.

Various methods can be used to construct the network given the set of selected solutions. Most methods have two basic components: (1) a scoring metric which discriminates the networks according to their quality and (2) a search algorithm which searches over the networks to find the one with the best scoring metric value. The BOA can use any scoring metric and any search algorithm.

In our recent work we used a simple greedy algorithm to construct the network given the data. In each iteration of the algorithm, the graph operation that improves the network score the most is performed. The simple operations that can be performed on the network include edge additions, edge reversals, and edge removals. Only operations that keep the network acyclic are allowed and the number of parents of each node can be bound by a constant in order to avoid superfluously complex models. The construction finishes when no operations are allowed or no applicable graph operation improves the score.

The next two sections briefly discuss the Bayesian-Dirichlet and minimum description length metrics that can be used to evaluate competing networks.

3.1 BAYESIAN-DIRICHLET METRIC

The Bayesian Dirichlet (BD) metric (Heckerman et al., 1994) combines the prior knowledge about the problem and the statistical data from a given data set. The probability of a Bayesian network B given data D can be computed by applying Bayes theorem as

$$p(B|D) = \frac{p(B)p(D|B)}{p(D)}. \quad (1)$$

The higher the $p(B|D)$, the more likely the network B is a correct model of the data. Therefore, the value of $p(B|D)$ can be used to score different networks and measure their quality. This measure is called a Bayesian scoring metric, or the *posterior* probability of B given data D. Since we are only interested in comparing different networks (hypotheses) for a fixed data set D, we can eliminate the denominator $p(D)$ of the above equation.

The probability $p(B)$ is called the *prior* probability of the network B and it can be used to incorporate prior information about the problem by assigning higher probability to the networks confirming our intuition or expert knowledge. By using an empty prior network (with no edges) the metric favors simpler networks (see Section 4.2). Under a number of assumptions, the following closed expression can be derived

for $p(D|B)$ (Heckerman et al., 1994):

$$p(D|B) = \prod_{i=0}^{n-1} \prod_{\pi_i} \frac{\Gamma(m'(\pi_i))}{\Gamma\left(m'(\pi_i) + m(\pi_i)\right)} \prod_{x_i} \frac{\Gamma\left(m'(x_i,\pi_i) + m(x_i,\pi_i)\right)}{\Gamma(m'(x_i,\pi_i))}, \quad (2)$$

where the product over π_i runs over all instances π_i of the parents Π_i of X_i, and the product over x_i runs over all instances x_i of X_i. By $m(\pi_i)$, the number of instances in D with Π_i instantiated to π_i is denoted. When the set Π_i is empty, there is one instance of Π_i and the number of instances with Π_i instantiated to this instance is set to N (the size of the data set D). By $m(x_i,\pi_i)$, we denote the number of instances in D that have both X_i set to x_i as well as Π_i set to π_i. The metric computed according to the above equation is called the *Bayesian-Dirichlet metric*, since one of the assumptions made to compute the formula is that the parameters are distributed according to a Dirichlet distribution.

Terms $m'(x_i,\pi_i)$ and $m'(\pi_i)$ express our beliefs in frequencies $m(x_i,\pi_i)$ and $m(\pi_i)$, respectively, and can be used as another source of prior information. A simple prior for the parameters $m'(x_i,\pi_i)$ and $m'(\pi_i)$ is to assume $m'(x_i,\pi_i) = 1$ for all x_i and π_i, and compute $m'(\pi_i)$ according to the above assignment. The metric using this assignment is called the K2 metric.

3.2 MINIMUM DESCRIPTION LENGTH METRIC

A minimum description length metric is based on the philosophical rule called Occam's razor, claiming that the simplest of competing theories be preferred to the more complex ones. The MDL metric favors short models in terms of their description length. A total description length of a data set D compressed according to a given model is defined as the sum of the space, measured in bits, required by the model, its parameters (various frequencies), and the data compressed according to the model. In context of evolutionary optimization the minimum description length was first time used by Harik (1999) in the extended compact genetic algorithm.

A directed acyclic graph can be encoded by storing a set of parents of each node. The set of parents of a particular node can be encoded by the number of the parents followed by the index of the set of parents in some agreed-upon enumeration of all possible subsets of variables of the corresponding cardinality. This results in $\log_2 n + \log_2 \binom{n}{|\Pi_i|}$ bits for the parents of X_i.

To store the conditional probabilities according to the distribution encoded by the network, we need to store all combinations of all but one values x_i of each variable X_i and all possible instances π_i of its parents Π_i. For each such combination of x_i and π_i the corresponding conditional probability $p(x_i|\pi_i)$ must be stored. For binary variables, there are $2^{|\Pi_i|}$ possible combinations of values of the variable and its parents (excluding one value x_i for each π_i, e.g. $x_i = 1$, for which $p(x_i|\pi_i)$ can be computed from the remaining conditional probabilities). This is an upper bound and can be reduced by using more sophisticated data structures to encode the conditional probability tables. To accurately encode each conditional probability, we can use $\frac{1}{2}\log_2 N$ bits (Friedman & Yakhini, 1996). Thus, the overall number of bits needed to store the table of conditional probabilities for X_i is $\log_2 N 2^{|\Pi_i|-1}$.

The number of bits needed to store an instance with some probability is given by a logarithm of this probability. We must sum the description lengths over all individuals in the population. The total length of the model, its parameters, and the data set compressed according to this model is then given by the sum of the above terms (Pelikan, Goldberg, & Sastry, 2000).

A major advantage of the MDL metric is that it favors simple models so that no upper bound on the model complexity has to be specified. This bound comes up naturally. However, when using a greedy algorithm for model construction, the problem of finding a valid model can become more difficult. Moreover, the MDL metric does not easily permit the use of prior information about the problem. In many real-world problems the utilization of expert knowledge (which is often available in some form) may be unavoidable. Section 4.2 presents another way of dealing with the complexity of models by specifying the prior probability of each model inversely proportionally to its complexity.

A similar metric, called the Bayesian Information Criterion (BIC), was used in the EBNA (Etxeberria & Larrañaga, 1999) and the LFDA (Mühlenbein & Mahnig, 2000) algorithms.

4 DECISION GRAPHS IN BAYESIAN NETWORKS

Instead of encoding the conditional probability tables by a simple but inefficient probability table, one can use more sophisticated structures such as decision trees, decision graphs, and default tables (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999). In this fashion the number of parameters required to fully encode the distribution can be signif-

icantly decreased and the models become more expressive. This section explains how decision graphs can be used to improve the expressiveness of Bayesian networks and the learning of Bayesian network structure. It provides a metric for computing the likelihood of Bayesian networks with decision graphs and a method for constructing such structures (Chickering, Heckerman, & Meek, 1997). The network construction algorithm of Chickering et al. (1997) takes an advantage of using decision graphs by directly manipulating the network structure through the graphs. This is a major difference from the way the decision trees/graphs are usually used in context of the MDL metrics to reduce the description length (Friedman & Goldszmidt, 1999). Additionally, we provide the assignment of prior probabilities which takes into account model complexity.

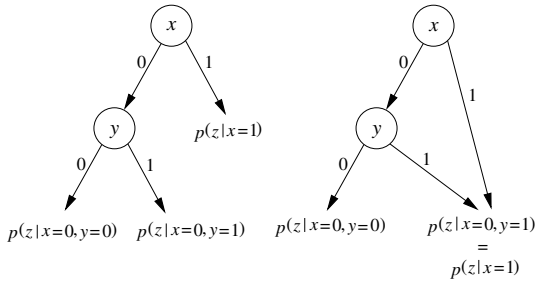


Figure 1: An example decision tree and graph encoding $p(z|x,y)$.

4.1 DECISION TREES AND GRAPHS

A decision tree is a directed acyclic graph where each node except for one designated node called the root has exactly one parent. The root has no parents. Non-leaf nodes of the tree are labeled by a variable (feature) on which we want to split. When a node is labeled by a variable v, we say that this node is a split on v. Edges from a split on v to its children (successors) are labeled by non-empty distinct exhaustive subsets of possible values of v.

To traverse a tree given an assignment of all the variables, we start in a root and on each split on v we continue to the child along the edge which contains the current value of v. Notice that for each instance (an assignment of all the variables) there exists only one possible way of traversing the tree to a leaf.

We will have one decision tree for each variable. Each leaf of this decision tree will contain conditional probabilities of different values of this variable given that the variables are constrained according to the path from the root to the leaf. An example adopted from Chickering et al. (1997) of a decision tree that encodes the conditional probability distribution $p(z|x,y)$ is shown in Figure 1. All variables in this figure are binary and thus we can split only to two children, one for 0 and one for 1. Instance $(x = 1, y = 1, z = 0)$ would traverse the tree to the right-most leaf. Instance $(x = 0, y = 1, z = 0)$ would result in the middle leaf.

A decision graph is an extension of a decision tree in which each non-root node can have multiple parents. By a decision graph, any set of equality constraints among conditional probabilities can be encoded. This can be shown by simply constructing a complete tree and merging all leaves that are equal. An example of a decision graph is shown in Figure 1. This decision graph can be obtained by merging the leaves $p(z|x = 0, y = 1)$ and $p(z|x = 1)$ which represents another equality constraint. It is important to note that the equality constraints, in fact, represent independence constraints. Moreover, each leaf in the decision graph for a variable represents independence assumptions of any variable not contained in the path from the root to this leaf, given the constraints specified by the corresponding path to this leaf. In fact, we do not need the Bayesian network anymore once we have the decision graphs. The network can be reconstructed by using the graphs.

There are four major advantages of using decision graphs in learning Bayesian networks. First, many fewer parameters can be used to represent a model. This saves memory and time requirements of both model construction as well as its utilization and allows representation of high-order relationships by reasonably-sized models. Second, the use of decision graphs allows learning a more complex class of models, because the relationships in a model can be cyclic under the constraint that different parts of the cycle are inconsistent with each other. It is beyond the scope of this paper to discuss this issue. Third, the construction of a Bayesian network with decision graphs performs smaller and more specific steps which may result in better models with respect to their likelihood. Finally, the network complexity measure can be easily incorporated into the scoring metric. The resulting measure allows the use of prior information unlike the MDL metric, and is as robust as the MDL metric when no such information is used. We will discuss this topic shortly.

4.2 BAYESIAN SCORE FOR NETWORKS WITH DECISION GRAPHS

In this section we briefly discuss the computation of a Bayesian score for Bayesian networks where conditional probabilities and independence assumptions for each variable are encoded by decision graphs (Chickering et al., 1997). This computation does not differ much from traditional Bayesian networks (see Equation 2). The outer product from Equation 2 remains the same. The middle product runs over all leaves of the decision graph G_i corresponding to the variable X_i. The inner-most product runs over all possible instances of the variable X_i. Thus,

$$p(D|B) = \prod_{i=0}^{n-1} \prod_{l \in L_i} \frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))} \prod_{x_i} \frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))}, \quad (3)$$

where L_i is the set of leaves in the decision graph G_i for X_i, $m(i,l)$ is the number of instances in D which end up the traversal through the graph G_i in the leaf l, $m(i,l)$ is the number of instances that have $X_i = x_i$ and end up the traversal of the graph G_i in the leaf l, the $m'(i,l)$ represents our prior knowledge about the value of $m(i,l)$, and $m'(x_i,i,l)$ represents our prior knowledge about the value of $m(x_i,i,l)$.

To adjust the prior probability of each network according to its complexity, we first compute the description length of the parameters required by the networks. To encode one frequency in the data set of size N, it is sufficient to use $0.5 \log_2 N$ bits (Friedman & Goldszmidt, 1999). Therefore, to encode all parameters, we need $0.5 \log_2 N \sum_i |L_i|$ bits, where $\sum_i |L_i|$ is the total number of leaves in all decision graphs. To favor simpler networks to the more complex ones, we can set the prior probability of a network to decrease exponentially with the description length of the set of parameters they require. Thus,

$$p(B) = c 2^{-0.5 \log_2 N \sum_i |L_i|}, \quad (4)$$

where c is a normalization constant required for the prior probabilities of all networks to sum to 1. The value of a normalization constant does not affect the result, since we are only interested in relative comparisons of networks and not the absolute value of their likelihood. A similar assignment of prior probabilities was presented in Friedman and Goldszmidt (1999). As we will see in the next section, the assignment in the last equation is sufficient to bias the model construction to networks with less parameters and avoid superfluously complex network structures without having to determine the maximal number of incoming edges in advance. This eliminates another degree of freedom for setting the parameters of the algorithm and thus makes the algorithm easier to use. Somewhat weaker pressure toward simpler networks was introduced in Heckerman et al. (1994) and Chickering et al. (1997). Our experience was that the latter pressure was not strong enough to result in efficient learning in our application.

The above assignment can be extended or fully replaced by the one that takes into account our prior knowledge about the problem by favoring models that are more similar to the prior network.

4.3 LEARNING BAYESIAN NETWORKS WITH DECISION GRAPHS

To construct a decision graph on binary variables, two operators are sufficient. The first operator is a *split*, which splits a leaf on some variable and creates two new children of the leaf, connecting each of them with an edge associated with one possible value of this variable, in our case, 0 or 1. The second operator is a *merge*, which merges two leaves into a single leaf. It does not make sense to split a leaf on a variable that was encountered on the path from the root to this leaf and therefore these operators will not be allowed.

For variables that can obtain more than two values, two versions of the split operator can be considered: (1) a complete split which creates one child for each possible value of the variable (as above), and (2) a binary split, which creates one child corresponding to one particular value and another child for all the remaining values. Other alternatives can also be considered.

The greedy algorithms for constructing a Bayesian network using decision graphs differs from the one presented in Section 3 in that it does not manipulate the constructed network directly but only by modifying the decision graphs corresponding to each variable. The decision graph G_i for each variable X_i is initialized to a single-leaf graph, containing only probabilities $p(X_i)$.

In each iteration, all operators (e.g., all possible merges and splits) that can be performed on all decision graphs G_i are examined. The operator that improves the score the most is performed on the corresponding decision graph. Both split and merge operators can be performed. When making a split, we must make sure that no cycles appear in the network B. To guarantee that the final network remains acyclic, we can continuously update the network B each time we perform a split. Once we split a leaf of the graph G_i on a vari-

able X_j, we add an edge (X_j, X_i) to the network B. A more sophisticated algorithm is possible which would allow a cycle whose different parts would be incompatible and thus not form a true cycle. The pseudocode of the above algorithm follows.

(1) Initialize a decision graph G_i for each node X_i to a graph containing only a single leaf.
(2) Initialize the network B into an empty network.
(3) Choose the best split or merge that does not result in a cycle in B.
(4) If no improvement is possible, finish.
(5) Execute the chosen operator.
(6) If the operator was a split, update network B.
(7) Go to (3).

It is important to notice the difference between the algorithm that directly modifies the network and the one which modifies the decision graphs. Adding an edge into a Bayesian network and using a full conditional probability table to store the corresponding probabilities corresponds to splitting *all* leaves of the decision graph corresponding to the terminal node of the edge on the variable corresponding to the initial node of the edge. However, by modifying only the decision graph, finer steps can be performed which may positively affect the quality of the resulting model.

5 EXPERIMENTS

This section starts by specifying our experiments. Subsequently, it provides and discusses the obtained results.

5.1 SPECIFICATION OF EXPERIMENTS

We have performed experiments on a number of functions. In this paper we only present results of our experiments on some of the functions. A simple linear function, called one-max, simply sums all bits. The 3-deceptive function is a sum of subfunctions of order 3, applied to disjoint blocks of 3 consecutive bits in the input string. Each of these subfunctions is defined as

$$f^3_{dec}(X) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

where X is a vector of 3 binary variables, and u is the sum of the input variables. The 3-deceptive function has one global optimum in $(1, 1, \ldots, 1)$ and $2^{\frac{n}{3}}$ local optima in all points where $(X_{3i} + X_{3i+1} + X_{3i+2}) \in \{000, 111\}$ for all $i \in \{0, \ldots, \frac{n}{3}\}$. The above function is deceptive in a sense that an average value over all strings that contain 0's in two particular positions is greater than the corresponding average with the 1's in these two positions. This feature makes the function very difficult unless the algorithm uses a recombination that respects interactions.

The two dimensional Ising spin-glass function maps bits onto a regular 2D grid. There are two types of edges. An edge of the first type contributes to the overall fitness by 1 if the bits at its ends are the same. An edge of the second type contributes to the fitness if the bits at its ends are different. For a more detailed description of the function and the definition of the problem instance we used in our experiments, please see Pelikan et al. (1998).

We have compared the simple GA, the BOA with both the MDL metric as well as the Bayesian-Dirichlet metric, and the BOA using decision graphs to construct the model and encode its parameters. The complexity measure was incorporated into the metric for the use with decision graphs as described in the above sections. The performance of the BOA does not depend on how we order the variables in solution strings. On the other hand, the ordering of the variables strongly influences the performance of the simple GA. We have chosen the representation so that interacting variables are close in the solutions strings. This is the best case for the simple GA. However, the main purpose of our experiments was not to compare the BOA to the simple GA but to give insight in the BOA and the effect of using local structures on its performance.

The population size in each algorithm was set as the minimal population size required for the algorithm to converge in all of 30 independent runs. Binary tournament selection was used in all experiments. The number of offspring is equal to a half of the population size and the generated offspring replace the worst half of the original population. In this fashion, the runs are more stable and elitism is introduced. The probability of crossover in the simple GA was determined according to empirical evidence presented elsewhere (Pelikan, Goldberg, & Cantú-Paz, 1999) as $p_c = 1$. Except for the one-max function, the mutation was not used, since it seems to not pay off on the tested problems. On one-max problem, $p_m = 0.01$ was used.

5.2 RESULTS

The results on the simple linear one-max function are shown in Figure 2. One-max is a linear function and thus it is expectable that all the algorithms perform very well. The BOA with an optimal $k = 0$ (which is equivalent to the UMDA) performs the best, because it does not take into account any interactions and in this problem all the variables are indeed independent.

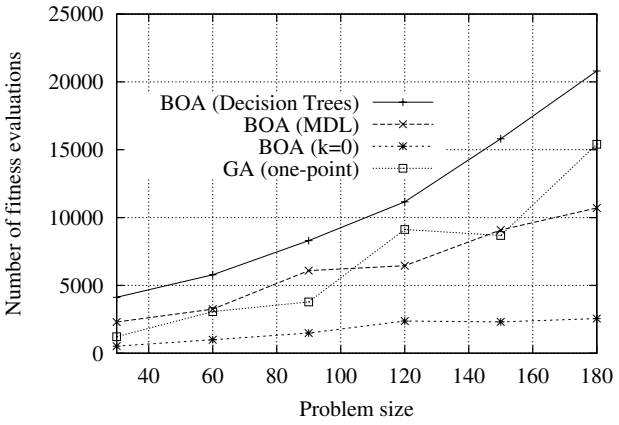


Figure 2: Results on the one-max function.

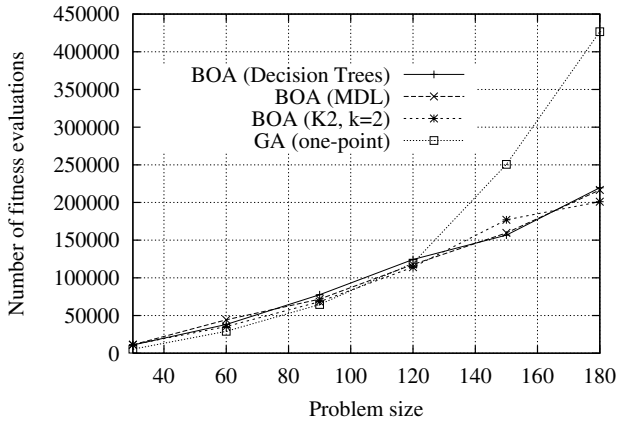


Figure 3: Results on the 3-deceptive function.

Other algorithms try to model some interactions and that is why they perform worse. However, all algorithms seem to scale very well. Using decision graphs does not seem to pay off in this case because of that the noise misleads the model building which is very sensitive when the decision graphs are used. Other modeling techniques make bigger steps and it is harder to mislead them. Even though this feature is likely to make the BOA work less efficiently on this simple linear problem, it can be very useful when solving difficult problems where making little steps while building the model pays off.

The results on the 3-deceptive function are shown in Figure 3. The results suggest that all versions of the BOA perform similarly and as the problem size grows, they outperform the simple GA with one-point crossover. This suggests that they scale up better. It is important to note that the coding chosen for this problem is the best one for the simple GA. If the building blocks (the bits corresponding to each deceptive subfunction) would be coded more loosely, the performance of the simple GA would get worse, and, eventually, for a random ordering of the variables in the strings representing solutions the simple GA would require exponential time (Thierens & Goldberg, 1993). However, the BOA is independent of the ordering of the bits in strings, and thus its performance would remain the same independently of the ordering we choose. The BOA with both the decision graphs and the MDL metric performs very similarly. This behavior is very interesting because the metrics are coming from two different paradigms.

The results on the spin-glass system are provided in Table 5.2. The BOA with the MDL metric performs the best. The BOA with $k = 2$ performs the worst. The simple GA does not reach the optimum even with huge populations and a large number of generations. This problem is very difficult for the simple GA. It is interesting that the MDL metric performs the best and outperforms both the BOA with decision graphs as well as a bound on the network complexity $k = 2$ and the K2 metric. We have also performed experiments with higher values of k. As the k increases to 4, the performance of the BOA with the K2 metric improves. The BOA with $k = 4$ and the K2 metric performs the best of all compared algorithms. However, with $k = 5$ the performance again decreases. This suggests that the value of $k = 4$ is a good choice for this problem.

Table 1: Results on the 2D spin-glass.

Algorithm	Fitness evals	Std. dev.
BOA (k=2, K2)	75833.33	4649.03
BOA (k=3, K2)	42733.33	2993.48
BOA (k=4, K2)	36606.67	2045.04
BOA (k=5, K2)	48960.00	2099.85
BOA (MDL)	38091.67	2317.69
BOA (Dec. trees)	57960.00	3109.62

6 CONCLUSIONS

The use of local structures to represent conditional probability tables has four major advantages. First, the number of parameters required to store probabilities with a large conditional part can decrease significantly. This makes the method work more efficiently as we increase the complexity of models. Moreover, high-order interactions can be represented by using models of reasonable size. Second, by using decision graphs to guide the network construction, one can discover more complicated relationships which may not be evident when directly modifying the network. Additionally, models of the same quality can be discovered by using less effort. Third, the models become more expressive, since cycles with different parts corresponding to in-

compatible paths in the decision graphs may appear in the model.

Finally, the complexity of the models can be automatically controlled by making prior probabilities of competing models be inversely proportional to their complexity. Our experiments suggest that setting the prior probability of a network to be inversely proportional to the number of bits required to store the frequencies in the network works well. By using Bayesian scoring metric containing a complexity measure as described above, one can both (1) use prior knowledge about the problem in network construction and (2) eliminate the need for a bound on the network complexity. In this fashion one can get the best of the two approaches.

Our test problems do not benefit from the compact representation by local structures, such as decision graphs. They only require covering interactions of a bounded order. Hot candidates for the use of local structures are problems that require a hierarchical approach where we must encode interactions of a very high order with a quite regular and simple structure. Such interactions may appear later in the run and require efficient representation.

ACKNOWLEDGMENTS

The authors would like to thank David Heckerman, David Chickering, Martin V. Butz, Erick Cantú-Paz, Josef Schwarz, and Jiri Ocenasek for useful comments and help with the project.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under a grant F49620-00-1-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Martin Pelikan was partially supported by grant VEGA 1/7654/20 of Slovak Grant Agency.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U.S. Government.

References

Bosman, P. A., & Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In Wu, A. (Ed.), *Workshop proceedings of the Genetic and Evolutionary Computation Conference GECCO 2000* (pp. 197–200). San Fransisco, CA: Morgan Kaufmann.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Etxeberria, R., & Larrañaga, P. (1999, March). Global optimization using Bayesian networks. In *Second Symposium on Artificial Intelligence (CIMAF-99)* (pp. 332–339). Habana, Cuba.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (1 ed.). (pp. 421–459). Cambridge, MA: MIT Press.

Friedman, N., & Yakhini, Z. (1996). On the sample complexity of learning Bayesian networks. In Horvitz, E., & Jensen, F. (Eds.), *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)* (pp. 274–282). San Francisco: Morgan Kaufmann Publishers.

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Heckerman, D., Geiger, D., & Chickering, M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.

Mühlenbein, H., & Mahnig, T. (2000). Evolutionary algorithms using marginal distributions. *New Generation Computing*, *18*, 157–166.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Eiben, A., et al. (Eds.), *Parallel Problem Solving from Nature - PPSN IV* (pp. 178–187). Berlin: Springer Verlag.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, California: Morgan Kaufmann.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1998). *Linkage problem, distribution estimation, and Bayesian networks* (IlliGAL Report No. 98013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W., et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I (pp. 525–532). San Fransisco, CA: Morgan Kaufmann.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*. In press.

Pelikan, M., Goldberg, D. E., & Sastry, K. (2000). *Bayesian optimization algorithm, decision graphs, and Occam's razor* (IlliGAL Report No. 2000020). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.

A Hybrid Approach to Support Interactive Data Mining

Josiah Poon
Basser Dept of Computer Science
University of Sydney
Sydney, NSW 2006
Australia
josiah@cs.usyd.edu.au
+61-2-9351 4920

Sham Prasher
School of CSEE
University of Queensland
St Lucia, QLD 4072
Australia
sham@csee.uq.edu.au

Abstract

Traditional data mining methodologies rely on a system-based objective function to obtain results that are considered interesting and accurate. The focus on computation speed and accuracy has neglected the domain user, who could contribute useful input to the decision-making of the mining process. In this paper, we outline, IMiN, a data mining architecture which has two specific goals. Firstly, an interactive and iterative framework based on the evolutionary model is proposed for the mining process to enable a domain user to control the flow of the mining process ad-lib. Secondly, this hybrid system maximizes the best from two different algorithms: Apriori algorithm and genetic algorithm. The former one is a proven data mining technique to zoom in and to construct interesting rules according to a predefined fitness function, whereas the latter one enables the search of interesting regions for further investigation. The results showed that the environment we constructed allowed the user to affect the mining direction during the mining process as they intended. The results also suggested that including Apriori in the process allowed the user to home in faster to solutions than normal. Overall IMiN proved itself capable of supporting the user's decision making during the mining process.

1 INTRODUCTION

In the past fifteen to twenty years there have been large developments in data mining techniques. Originally the emergence of simple algorithms has paved the way for new and innovative techniques that use smart techniques to analyze information. Methods proposed in (Bayardo Jr., 1998) and (Yip *et al*, 1999) take the approach of cleverly deriving new useful information from previously successful knowledge. These techniques use dependency modelling whereby a relationship between two or more elements is defined by how strongly the grouping of one, or more, is a reliable predictor of the others. Another technique that has been used to model such relationships is genetic algorithms. The rationale with this approach is that finding useful information is synonymous to searching for a problem solution. Typical examples in this category include (Marmelstein, 1998) and (Noda *et al*, 1999).

The afore-mentioned research is lacking in the sense that knowledge discovery has become largely autonomous and hence distant from its original focus. Many algorithms leave out the capacity for human input and attempt to mimic a semantic level of understanding of the information using objective functions and algorithms. Current methods that attempt to include the domain user into the mining process are still limited and have yet to reach their full potential.

The purpose of paper is to propose a data mining system that will allow user to steer the direction of the mining process as it happens. This will be facilitated by the complementary strength of the exploration power of genetic algorithm and the dependency analysis of a well-established data mining techniques. The rationale for using a mixture of techniques follows the principle that biases can be removed by combining one paradigm with another that balances aspects of the original paradigm (Anand and Hughes, 1998). We plan to take those aspects that are useful to us from each method and combine them in a way that makes them compatible. The system is also equipped with an interface to enable interaction. The rationale behind our approach agrees with the statement (Williams, 1999) that "… data mining is an inherently *interactive* and *iteractive* process …", meaning that the system should act as a support mechanism to the user's intuitions.

In the next section, we outline in more detail the background of this paper. In section 3, we propose the design of, IMiN, an interactive data mining system. The experiment and results are outlined in section 4. This is followed up with final conclusions and future recommendations in section 5.

2 RELATED WORK

In this section we overview various areas of previous research that are related to the direction of this paper. The main areas of focus are itemset building algorithms, genetic algorithms, and the development in interactive mining.

Agrawal and Srikant (1994) started the family of Apriori algorithms. This family of algorithms is incremental in nature. The algorithm first scans the dataset and builds a list of itemsets L_1, which lists the support of all single element itemsets. In each subsequent iteration, the algorithm builds a set C_k containing large itemsets generated by joining fit ones (satisfying a minimum support requirement) from the previous iteration L_{k-1}. Then, a function called apriori-gen is called to prune all itemsets in C_k that are deemed unfit.

Instead of re-checking the dataset to ascertain the fitness of itemsets in C_k, apriori-gen uses a smart heuristic. An itemset is said to be unfit if not all of its L_{k-1} subsets are fit. Once these itemsets are pruned, the dataset is scanned to determine the support of the remaining itemsets in C_k and hence the process repeats again. The entire process keeps running until no more itemsets can be generated from the current ones. To illustrate apriori-gen in action consider the following example:

Let L_3 consist of {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}. After the join step the output will be {{1 2 3 4}, {1 3 4 5}. Pruning this with Apriori-gen will result in {1 2 3 4} because each of its subsets {1 2 3}, {1 2 4}, {1 3 4}, {2 3 4} exist in L_3.

To summarise, itemsets are built upon previously constructed ones that are considered fit. Unnecessary computation is avoided on those itemsets that are inappropriate. These in turn produce specific associations. Since this algorithm was introduced, there has been a steady advance in the smart generation of itemsets.

More recent work such as the LGen technique (Yip *et al*, 1999) improves on Apriori by generating variable-sized itemsets on each pass of the dataset. The goal is to reduce the overall I/O passes required to find all large itemsets. The size of the largest itemset in Apriori algorithms dictates the number of passes through the data that are required. LGen, on a best case scenario, can require as few as two passes irrespective of dataset size. Another lattice-based algorithm Max-Miner (Bayardo Jr., 1998) uses a look-ahead technique whereby large itemsets are generated earlier than they would be in Apriori. These methods, while being at the forefront of their paradigm, generally lack the structure for interaction.

Genetic algorithms are a newer paradigm in data mining and have proven compatible performance against other methodologies. GRaCCE (Marmelstein, 1998) is a genetic rule induction algorithm that gradually refines linear boundaries drawn on the data to deduce rules. GRaCCE was tested against CART, a decision tree induction algorithm, and was found to generate fewer and more concise rules than CART. The results demonstrate that the searching capabilities of genetic algorithms can be applied successfully and efficiently to data mining in a rule induction context.

In a different study (Noda *et al*, 1999) a genetic algorithm is used in a dependence-modelling task to discover interesting prediction rules. An algorithm was proposed to combine some characteristics of the GA-Nuggets algorithm (Freitas, 1999) and the objective evaluation of rule interestingness (Freitas, 1998). A noteworthy difference between this and rule induction is the latter requires some preconception of the desired result whereas the former is more free to discover surprising information.

The focus of the Evolutionary Hot Spots model (Williams, 1999) is that: data mining is an inherently interactive and iterative process. The Hot Spots model uses induction and clustering techniques to identify potential groups of interesting rules. These groups are then evaluated for interestingness. The Hot Spots process is worked into an iterative architecture in order to evolve the definition of interestingness along with the mining process.

First the Hot Spots method defines interestingness based on unexpectedness or surprising characteristics. The database, D, consists of a set of entities, where each one is a tuple. Hot Spots generates a set of rules R = {r_1, r_2, …, r_p} where each rule holds a group of entities. Each rule is also known as a nugget. The problem is that the number of nuggets that arise in very large databases can itself be vast, many of which can be expected to have little interesting quality. For this reason an evolutionary approach was adopted to refine the quality of produced rules. Also meta conditions are implemented to limit the number of rules that are presented to the user.

Initially rules are found using some statistical rating, like a simplified version of the genetic algorithm fitness function. A number of nuggets *q*, each having approximately *s* rules, is discovered and presented to the user, who will rate the nuggets and feedback into the system a modified definition of interestingness *q*. At each iteration the user evaluates discovered rules and further evolves these measures of interestingness.

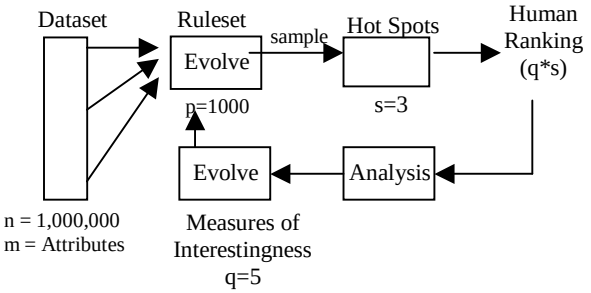


Figure 1 – The Evolutionary Hot Spots Model

The process is designed to stop at the discretion of users, not the system. The architecture (Figure 1) is an example where the user's input is taken not only at the

beginning but also throughout the mining process. The potential for interaction is an important aspect with regard to the aim of this paper.

3 IMiN – A HYBRID MODEL

Our proposed solution is an interactive system that both guides and supports the user's decision-making during the mining process. If we look at data mining as a search for good solutions, then the system should allow a user to search as broad (exploration) or as deep (exploitation) as is desirable.

The proposed hybrid approach combines the genetic algorithm developed by (Noda *et al*, 1999) with the Apriori itemset building algorithm (Agrawal and Srikant, 1994). The interactivity is made available by the incorporation of the Evolutionary Hot Spots architecture. This combination (Figure 2) provides us with a basic framework and technique to enable human interaction in KDD. It also provides us with the opportunity to test the augmentation of genetic algorithms to the Apriori family.

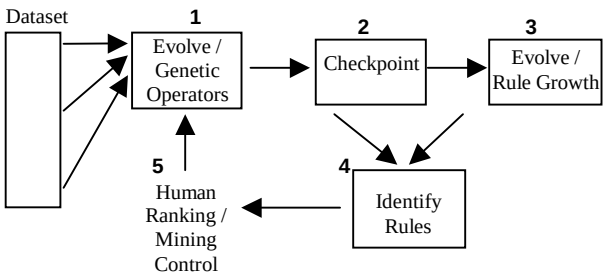


Figure 2 – Modified system architecture to accommodate the hybrid

An initial population of rules is generated and evolves for a certain number of generations in Step 1. Then the system checks whether it is supposed to further evolve rules by growing them (Step 2). Note that rule growth (Step 3) is synonymous with increasing the order of schema in a chromosome. In Step 4, rules that are fit according to the objective fitness function are displayed to the user and to be further manipulated (Step 5). The user selects which rules will continue subsequent evolution and then the cycle begins with null alleles which are marked with a '-1'.

3.1 THE ARCHITECTURE: STEP-BY-STEP

3.1.1 Step 1: Evolution of Rules

The genetic algorithm in this model searches for association between attributes in a dataset where a rule consists of an antecedent and consequent. The user first selects a group of attributes that are eligible to be goal attributes. In order to limit the search space, a small set of attributes that interests the user remains, and thus cuts down on the total processing time.

Rules in the system are represented by chromosomes. A chromosome's length corresponds to the number of attributes in the dataset. Only values of the antecedent are stored in the chromosome. This structure allows us to apply the genetic operations easily and take advantage of the efficient one-pass method 2. The consequent is stored externally. Specific details on the genetic operators will be covered in the following sections.

The exploration of the search space in GA is generally enabled through the crossover operator. Many techniques, including GA-Nuggets, use uniform crossover. However, it is unsuitable for the proposed system because the number of antecedents (or itemsets) can only be incremented at each rule growth in the Apriori algorithm. The construction of a *(k+1)*-order itemsets relies on the existence of the *k*-order, (*k-1)*-order, *(k-2)*-order, .., *2*-order, *1*-order itemsets. Chaotic fluctuation of antecedents between generations due to the random nature of genetic operators is not acceptable. Consider the example in Figure 3:

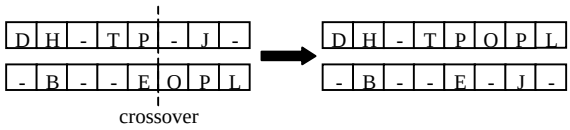


Figure 3 – Effect on schema order in 1-point crossover (null alleles are denoted by a '-')

In Figure 3 the offspring (right) have a mixture of features from the parents (left), hence the schemas' order has changed.

While GA-Nuggets attempts to regulate rule size to a certain degree using the specialised *insert* and *remove* operators, they do not guarantee the level of control necessary to make scheduled rule growth possible. To ensure that the rule size remains constant after crossover, a new crossover method termed *Random-Quota Crossover* (RQC) is introduced which is adapted from the 1-point crossover. First, a random number between 0 and the schema order-1 is chosen. This number represents a quota of the number of alleles that must be copied from each parent. Then, a brute-force pass is required to work through the parents from the head to the tail to swap the non-null alleles until the quota is reached.

Figure 4 shows an example of RQC. The quota is set to '3' in this example, which means only 3 alleles needed to be swapped with the mating partner. First of all, the first positions of each of the parents are checked for non-null values. Since parent-A contains a value in this position, the allele in this position is swapped with parent-B. After swapping, the number of swapped non-null alleles for parent-A and parent-B are 1 and 0 respectively. Then, the operator moves to the next position and finds that both parents have values in locus 2. The value of 'H' and 'B' are swapped between the two parents. On completion of the second locus, the numbers of non-null alleles swapped for parent-A and parent-B become 2 and 1. Since both parents have null values in their third gene,

no swapping takes place. For the fourth position, parent-A has a value 'T' in the current location. Swapping occurs and the numbers of swapped alleles in the two parents are 3 and 1 correspondingly. Note that 'E' and 'P' are not swapped between parent-B and parent-A. It is because when the algorithm reached that particular position, that quota for A has already been satisfied. As a result B must wait until it finds an empty allele in A to swap to. Both rules begin with the same number of non-null alleles, RCQ guarantees that the resulting rules have the same property as well.

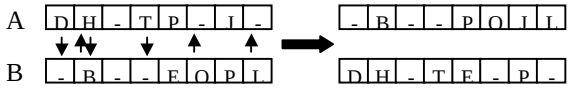


Figure 4 – RQC with quota set to 3

Each feature of every rule chromosome is given a percentage probability to mutate after crossover is performed. The chance of each feature to mutate is the same within the same generation, however, this may change in subsequent generations at user's discretion.

3.1.2 Step 2: The Checkpoint

At certain periods in the mining process, rules will be 'grown'. After individuals are evolved for a set number of generations, the system will proceed to rule growth using the Apriori technique.

3.1.3 Rule Growth

In this step, the order of antecedents in every rule is grown by one. In the subsequent discussion, we treat the rule antecedents as itemsets, where an itemset is a list of elements $A_{ij}…A_{nj}$ such that A_i is the ith attribute and A_{ij} is the jth value of this attribute. The support of an itemset denotes how often all of its elements occur together. During the rule growth, we are concerned with the support of itemsets, not the associations derived from them. The antecedents of associations produced by the genetic operators are reverted to itemset form. From then on, the Apriori is applied. After rule growth, new associations are then extracted from the grown itemsets.

L_{k-1} is obtained by collecting all the antecedents of associations that have sufficient support. An example of Apriori growing itemsets is shown in Figure 5. The guaranteed order allows us to reduce the amount of necessary comparisons. In our system the generation of L_{k-1} needs to be done only during the generation before rule growth occurs.

Due to genetic operations in our system, the ordering of chromosomes in a generation, if any, cannot be preserved. To deal with an unordered set of rules, the algorithm is modified to ensure that rules are only combined with other rules such that the last non-null value of the second chromosome occurs after that of the first. The process is described in Figure 6. Following this principle, we add 8 from A to B to produce D, 4 from C to B to produce E, and 8 from A to C to produce F. Chromosomes D, E and F constitute the unpruned C_k set. When C_k is created, all non-large rules are then removed by a pruning step, which is the latter part of apriori-gen.

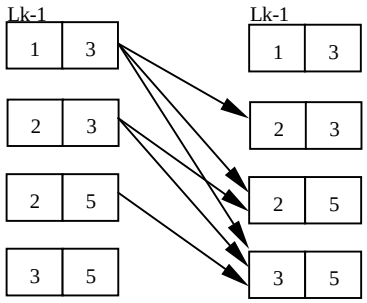


Figure 5 – Apriori ordered join

The difference between this modified version and the original Apriori algorithm is that: it is used only once in each rule growth stage, instead of repetitively until no more large itemsets are generated.

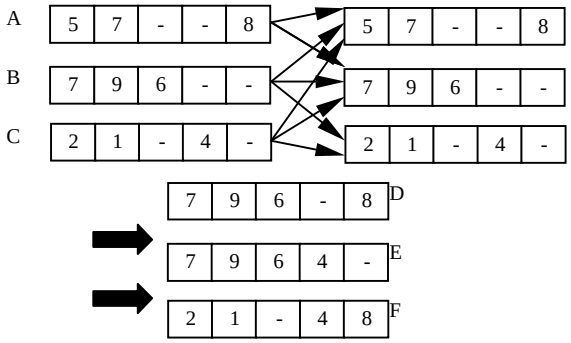


Figure 6 – Growth Example with Chromosomes A, B, C (L_{k-1}). To ensure that there is no redundancy chromosomes only grow by adding alleles from other chromosomes that occur after their own last non-null allele. An alternate approach would be to order the individuals then compare them as is done in Figure 5.

3.1.4 Step 4: Identifying Rules

After a number of generations, the system displays all rules in the population that satisfy the required support and confidence levels. This period is called the *generation increment* and represents how often a user interacts with the system. Because of the nature of genetic algorithms, certain rules may be duplicated within a population. The exhibition of rule dominance within a population is not uncommon. Hence, to ease the user's task of having to analyse the rules, all the duplicates are not displayed.

3.1.5 Step 5: Human Ranking and Parameter Adjustment

Once the system has completed a single cycle of rule evolution (Step 1), rules that satisfy minimum support and confidence are displayed to the user. Users can

manipulate the rules and/or mining parameters for subsequent rule evolution.

A subset of the produced rules can be selected to continue the mining process. In other words, the user selection represents the subjective part of defining interestingness within the system. When the user preference changes over time, this subjective fitness function is reflected in the population of rules that is developed in subsequent generations. Hence, the user complements the objective fitness function. The implication of selecting a subset of the original population is to speed up the convergence of the population. This acceleration will be amplified if the search is further narrowed down when the user becomes more confident about the mining goals.

Alternatively, if the user has little direction, they may continue to investigate all the rules the system produces. This will allow the genetic algorithm to search through an untouched and still relatively large search space with a diversified population. As discussed above, the speed of convergence is related to diversity within the population.

User can also customise the mining process in Step 5 by adjusting the mining parameters. Changing the mining parameters means that the objective fitness function places a different emphasis on certain elements (eg confidence threshold, confidence weight) over others. The specific changes available to the user can vary depending on user preference.

In many situations, potential points in a search space are crowded and suffocated by other points that are closed to local optima. When a genetic algorithm faces this situation, it tends to converge prematurely, i.e. narrowing the scope of exploration. While this may represent a legitimate homing in on good solutions, it can also mean the exclusion of others. This is generally dealt with in GA using crowding, fitness sharing or disabling the mating of similar individuals to escape from the trap of local optima. Having said that, the local peaks in a search space may represent opportunities to the formulation of business strategy for niche groups. To help avoid this anomaly to occur, we introduce *extended parallel exploration*, which allows the user to artificially increase the fitness of a set of rules. Doing this will reduce the current domination of the population by an elite group of rules and will allow others a chance to live and to develop. Exactly how much the convergence is slowed down depends on how much the fitness of certain rules is increased. The greater the increase, the longer those rules will remain in the population. In essence, the increase only extends the life of certain rules so that they can be explored. What this function allows is a deviation from the system-controlled mining path to areas in the search space deemed interesting by the user.

4 EXPERIMENT AND RESULTS

This section describes the tests performed on the IMiN system. The purpose is to test the domain user's ability to control the directon of the mining process using IMiN. The system was tested with a dataset (*census income*) containing census information on US incomes. This dataset came from UCI repository. It contains 6 continuous and 9 categorical attributes. The dataset has 7% amount of noise in the form of unknown values.

4.1 TEST 1: CHANGING THE MINING FOCUS - USING SELECTION OF RULES

The purpose of this test is to illustrate the user's ability to change the focus of mining during the mining process by selecting rules to be evolved.

We first did a control run in which the system had total control over the mining process. During this run, all rules were selected for evolution. The control test run was used as a standard to compare the user-controlled run with. Parameters set for the control and user runs were:

Parameters:	population: 150, sample: 500, minimum support: 5, minimum confidence: 60%, crossover: 60%, mutation: 0.01%, support weight: 1, confidence weight: 2, small rule filter: 50
Goal Attributes:	age, workclass, education, marital status, occupation, relationship, hours-per-week, class
Continuous Attributes (Interval Size):	age (3), hours-per-week (3)

The initial population was allowed to evolve 15 generations. This was done ten times to get a generalised view of which rules were likely to appear. A superset of all the rules from these ten runs was prepared and it contained 86 unique[1] rules. The sizes of these populations ranged from 20 to 33 unique rules, with an average size of 26.1 rules from a total of 261 rules over the 10 separate runs. This meant there was a moderate amount of overlap between the ten runs. The superset represented a common set of rules that the system generally converged toward after 15 generations.

Rules are grown at this stage and developed for another 10 generations. We ran this five times and noted the rules produced at the end. The superset for 2-order rules[2] contained 97 unique rules from a possible 122. The difference between 2-order populations was much greater than that between 1-order populations. This phenomenon indicated that the search direction spreaded out more and

[1] A count of unique rules includes each rule only once. If a rule appears several times in the same population its duplicates are not counted. The 'population' or 'population size' used in this paper refers to the number of unique rules in a population.

[2] An *n-order* rule contains n elements in its antecedent. Similarly an *n-ordered / order n* population is a populations that contains only *n-order* rules.

became increasingly unpredictable. Also, some of the difference might be caused by hidden itemsets that only produced fit rules after rule growth. As a result, predicting the direction of mining was very difficult beyond this stage. In turn, this limited the extent to which we could compare the control run results with the user run results.

For the user-controlled run, the same random seed was used to generate the initial random population. This guaranteed that all tests began with the same population of rules. We selected a set of rules in the initial population and develop them for fifteen generations. The rules we investigated were those that had values from the *age*, *education*, or *hours-per-week* attributes as their antecedent. This was performed ten times and a superset of all the resulting rules was created.

The populations of the ten runs ranged from 3 to 16 rules in size. This superset contained 37 unique rules out of a total 95 from the ten runs. This set contained only rules that had antecedents containing values from the *age*, *education*, or *hours-per-week* attributes. The results shown that the focused rules started to dominate in subsequent generations. The final superset was a subset of the results from the 15th generation in the control run. This shown that we were able to focus on one area the GA originally considered, and discontinued the rest.

From the ten runs of both the control and user runs, the number of unique rules (population sizes) at each generation were noted and averaged. The result is shown in Figure 7. The figure illustrates the rates of convergence between the control run and the user-controlled runs. From the results we can see that the user runs converge faster because there is less variation within the population as a consequence of selection. This means there is less exploration in the user runs.

After the fifteenth generation, although there was a sharp increase (the spike) in population size in the control run, the convergence still carries on. Regarding the user runs only one out of ten produced any rules after rule growth, which evidently disappeared before another five generations passed. This indicates that the system had already converged to a maximum by the 17th generation.

4.2 TEST 2: USING EXTENDED PARALLEL EXPLORATION

While we may want to focus on certain areas, we may also want to continue investigating others. Normally we could do this by simply selecting all rules for further investigation. However, rules of interest may be later excluded from the population because they have a low objective fitness at that stage. The purpose of this test is to:

- Extend the search in an area that the system normally would not explore extensively.
- Maintain exploration of other areas while focusing on one in particular.

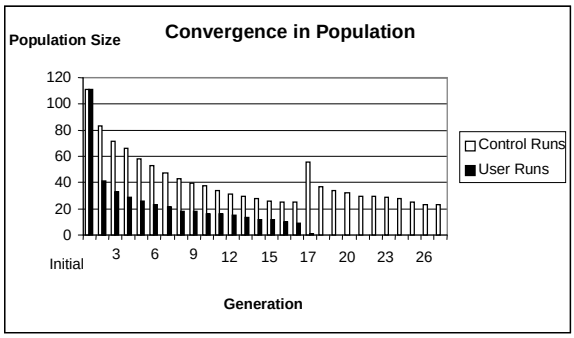


Figure 7 – Population size throughout mining

In this test, we planned to track the progression of rule discovery more finely than in the previous test. For this reason, another control test is generated. A fixed seed is used for all random generation of the initial population generation, crossover and mutation probabilities. This guarantees that the control run is completely predictable. Any changes that occurred in user runs could thus be easily spotted. We further tried this procedure with different seeds to ensure the results could be generalised.

In the first user run, we set the random seed to 2. We doubled the fitness of any rule with *age 48.0* in the antecedent in the initial population. In the second user run, the seed was set to 9. In the initial population we doubled the fitness of all rules with *education Doctorate* in the antecedents. In the control run of both cases, no rule with the specific paremter-value survived past the first generation. Fitness was not altered at any other time. No rule with this antecedent survived long in the control run. Parameters set for the control test and user-controlled runs were:

Parameters:	population: 50, sample: 500, minimum support: 5, minimum confidence: 60%, crossover: 60%, mutation: 0.01%, support weight: 1, confidence weight: 2, small rule filter: 50
Goal Attributes:	age, workclass, education, marital status, occupation, relationship, hours-per-week, class
Continuous Attributes (Interval Size):	age (3), hours-per-week (3)

Rules were evolved for six generations prior to rule growth then for another five generations after it. Figure 8 shown a comparison of population sizes between the control and user runs. Figure 9 shown the break-up of rules in the user controlled run. ‘Different rules’ are those rules that occur at a particular generation in the user run but do not appear in the same stage in the control run. Rules that appear in the control and user runs in the same generation are ‘same rules’. Hence the sum of the two

series shown in Figure 9 is the equal to population size of the user runs in Figure 8.

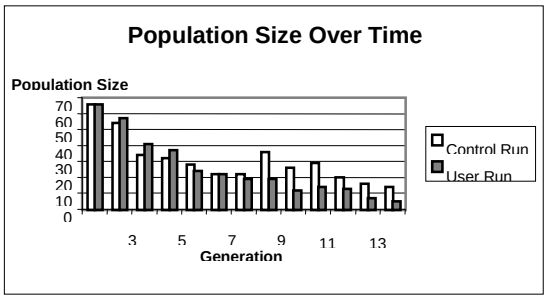


Figure 8 – Comparison of population size (seed=2)

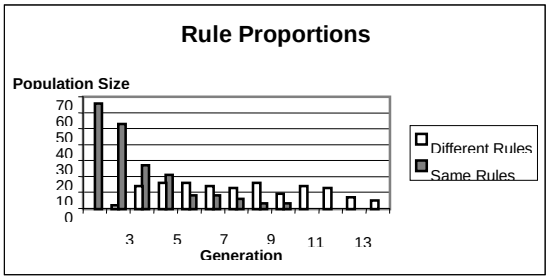


Figure 9 – Comparison of rules proportion (seed=2)

We could see from Figure 9 that after the 10th generation, none of the original rules from the control run remained in the user run. In the first user run, rules with *age 48.0* in their antecedent survived to the last generation. In the second user run, rules with *education Doctorate* in their antecedent persisted to the second last generation.

4.3 DISCUSSION

The results from Test 1 shows that the user is able to alter the focus of mining by selecting a subset of rules in a population. The effect of this change is a hastened convergence due to the loss of diversity in the population. Figure 7 illustrates the rate at which the population converged after user-intervention. It shows that exploitation is dominant over exploration. This represents a situation where the system is homing in on a particular area. The resulting supersets of the control and user runs show that the system changed its search focus in accordance with the user's intention.

We can see from Test 2 (Figure 9) that the number of 'different' rules increased over time, eventually surpassing the number of 'same' rules. This demonstrates the shift of focus from one set of rules to another. As mentioned earlier, this shift can partially be attributed to occurrence of rules at different stages in the mining process. The effect is stronger in the first user run because a larger proportion of rules had their fitness enhanced.

However, the number of different rules at each stage in the user run is far greater than the number of enhanced rules. This means that the population is sensitive to the fitness enhancements in subsequent evolution. When a rule's fitness is increased, the probability that it will be selected for crossover also increases. Thus rules that were originally selected in the control run at a particular stage were not in the user run. This explains why certain rules appear at different stages between the two runs. Despite these changes the system still maintained a focus on the original rules that were enhanced for most of the process. This shows that the user is able to focus on areas that they are interested in.

Our general observation on the control and user runs revealed that this change in foci may not be so dramatic. We discovered that many of the rules that were classified as different were simply discovered at a different generation than in the control run. Rules having partially different antecedents were also counted as 'different'. The areas explored in the user run were thus similar to those explored in the control run and the number of actual new/underived rules that were discovered in the user run was negligible.

Having said that, the system is able to change one or more of its mining foci in response to the user's intervention, and in accordance to the user's intended direction. These results show that the system is able to home in to the solutions faster with the aid of a user. Test 2 proved that the system allows a user to focus on an area of interest without having to lose track of other areas.

From the results (esp. Test 1), we see the hybrid system helped us to find solutions quickly. This is one advantage offered by the hybridization of Apriori and the GA. Also, users can now incrementally zoom into interesting regions while the system constructs more complex rules (itemsets) along the way.

5 CONCLUSIONS

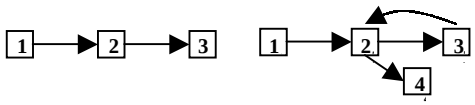
In this paper we presented a framework (IMiN) that uses evolutionary paradigm to augment the Apriori algorithm. This hybrid approach provides an exploratory tool to a large search space and enables the incremental construction of itemsets with proven data mining technique.

The testing of the system yielded promising results. We demonstrated the potential of a user interacting with, and steering, the mining process. The results of these tests proved that the system considered input from the user and changed the direction of knowledge discovery. The change also had strong bearing on the direction that was the user's intention. The ability to change the direction of mining during the mining process meant that the end results were likely to be more meaningful to the user than that produced from a purely system-oriented process. From this we can conclude that the interactive evolutionary architecture is beneficial to the mining process.

One observation we made while testing the system was that decision-making in this system is difficult

without a certain degree of retrospect. In our tests, the decisions came from analyzing the results of a control run. However, this does not exist in a real-world situation. Although the direction control allows more desirable results to be found, there is no way of undoing choices in the iterative process. Since one of the system's goals was to facilitate user-controlled exploration, finding ways to allow more robust and diverse searching is open for future work.

Regarding the difficulty of decision-making ad-lib, we propose to increase to searching capability of the system using 'backtracking'. If we look at the mining process as a series of user interaction stages between periods of system computation, then mining in the current system is a linear process. What we propose is giving the user the ability to backtrack in this linear sequence to a previous state of the mining process. This would allow them to then take a different direction at that point. This also means that decisions about exploration would not always need to be concrete since actions that yielded undesirable results could be undone.



(a) Linear (b) Non-Linear

Figure 10 – Structure of the mining process

Figure 10a shows the current nature of the mining process as a linear sequence of interaction stages. Ideally we would like to undo undesirable steps by backtracking to a previous stage and altering our course to obtain different results. Figure 10b illustrates this as the transition from 2 to 3, backtracking back to 2 and plotting a new course resulting in the discovery of new rules at 4.

Due to the nature of the system, testing has been quite limited and controlled in order to gather reliable results. The unpredictable nature of genetic algorithms makes large-scale empirical testing very difficult. However such tests can provide greater insights into the reliability of this system and are recommended. Also the determination of suitable mining parameters has long been an issue in the field of KDD. By introducing this new system we leave open the opportunity to conduct tests on the effects RQC has on the rest of the system, and hence the results. However such extensive testing is a study in itself and requires more attention than is possible in this paper.

References

Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules. *20th International Conference on Very Large Databases (VLDB-94)*, 487-498.

Anand, S. S. and Hughes, J. G. (1998). Hybrid Data Mining Systems: The Next Generation. In X. Wu, R. Kotagiri and K. B. Korb (ed.), *Second Asia-Pacific Conference, PAKDD-98*, 13-24. , Melbourne, Australia, Springer-Verlag.

Bayardo Jr., R. J. (1998). Efficiently Mining Long Patterns from Databases. In X. Wu, R. Kotagiri and K. B. Korb (ed.), *Second Asia-Pacific Conference, PAKDD-98*, 85-93. , Melbourne, Australia, Springer-Verlag.

Freitas, A. A. (1998). On Objective Measures of Rule Surprisingness. In X. Wu, R. Kotagiri and K. B. Korb (ed.), *Second Asia-Pacific Conference, PAKDD-98*, 1-9. , Melbourne, Australia, Springer-Verlag.

Freitas, A. A. (1999). A Genetic Algorithm for Generalized Rule Induction. *Advances in Soft Computing - Engineering Design and Manufacturing*, 340-353. , Springer-Verlag.

Marmelstein, R. E. (1998). GRaCCE: A Genetic Environment for Data Mining. In J. R. Koza (ed.), *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Stanford University Bookstore.

Noda, E., Freitas, A. A. *et al*. (1999). Discovering Interesting Prediction Rules with a Genetic Algorithm. *CEC-99*, 1322-1329.

Williams, G. J. (1999). Evolutionary Hot Spots Data Mining: An Architecture for Exploring for Interesting Discoveries. In N. Zhong and L. Zhou (ed.), *Third Asia-Pacific Conference, PAKDD-99*, 184-193. , Beijing, China, Springer-Verlag.

Yip, C. L., Loo, K. K. *et al*. (1999). LGen - A Lattice-Based Candidate Set Generation Algorithm. In N. Zhong and L. Zhou (ed.), *Third Asia-Pacific Conference, PAKDD-99*, 54-63. , Beijing, China, Springer-Verlag.

The effects of crossover and mutation operators on variable length linear structures

Jonathan E. Rowe
School of Computer Science
The University of Birmingham
Birmingham B15 2TT
J.E.Rowe@cs.bham.ac.uk
(+44) 121 414 2985

Nicholas Freitag McPhee
Division of Science and Mathematics
University of Minnesota, Morris
Morris, MN, USA
mcphee@mrs.umn.edu
(+1) 320 589-6300

Abstract

In the search space of variable length strings, it is possible to define crossover and mutation operators that are equivalent to those used in genetic programming on tree structures. We study the effects of these operators on the lengths of strings within a population. It is shown that the distributions by which different string lengths are sampled are strongly biased. To investigate these biases, the effects of repeated application of the operators (without regard for fitness) is considered, and in some cases the fixed-point distributions are found.

1 Introduction

We analyse the effects of three different operators on a population of variable length strings. These operators have previously been studied in [McPhee et al., 2001] using numerical simulations. In this paper we provide proofs of some of the observed behaviour.

The operators we are studying are purely syntactic, making no reference to fitness. Each operator is defined to act on a string in the following manner:

1. truncate the string randomly
2. concatenate another string on to the result

Strings are truncated by throwing away at least one element. The number of elements thrown away is chosen uniformly at random. The string to be added to the result of this process depends on which operator is being used. The three we will discuss are[1]

Crossover Choose another string from the population and truncate it (i.e. take a random suffix from it, including the terminal).

Full mutation Add a randomly generated string of a fixed length.

Growth mutation Add a randomly generated string of a random length, determined by a geometric distribution.[2]

In this study we shall use a generational model where the selected operator is applied to each string in the population in turn to give a new population. Selection from the population, where required, is done uniformly at random.

One may think of the first string as being a selected parent program. By truncating it, we are taking a random prefix. If programs are thought of as a sequence of unary functions which will be applied to the final string element (the terminal element), then the truncation throws away the terminal and possibly some of the functions. The end of the new offspring program (including a new terminal) comes from the second string that is concatenated with this prefix. It should be noted, however, that because we are not considering fitness-based selection (or, equivalently, considering a flat fitness function), that all we are really concerned about is the *number* of elements in a string, not their meaning.

We will analyse the effects of each operator by considering the truncation and concatenation stages separately. A population at a given time t will be represented by a random variable X_t representing the distribution over lengths that exist in the population. We will let T be the random variable denoting the length

[1] Versions of these operators defined for tree structures can be found in [Koza, 1992].

[2] This is more commonly referred to as the *grow mutation method*.

of strings after truncation. Some information regarding the distribution of T will be given in the following section.

For each operator we will have another random variable, C representing the lengths of the strings to be concatenated. The probability distribution for string lengths resulting from the application of an operator will be the distribution of the sum of T and the concatenation variable C. To assist in finding these distributions, we will use *probability generating functions* (pgfs). Given a random integer variable Z, the probability generating function (pgf) for Z is:

$$G_Z(z) = \sum_{k \geq 0} \mathbf{Pr}[Z = k] z^k$$

One of the advantages of using pgfs is that the pgf of the sum of two random variables is given by the product of the pgfs of the two random variables, that is

$$G_{T+C} = G_T G_C$$

The product of the two series gives the *convolution* of the two distributions [Graham et al., 1994].

For each of the three genetic operators mentioned above we will look at their effects in one generation on the mean and variance of the distributions of lengths represented by X_t. These effects can be calculated exactly. We will also consider the results of repeated application of the operators (i.e. over many generations) to gain an idea of how strongly they are biased towards particular fixed-point distributions. Strictly speaking, these results apply only in the infinite population limit, in which case the equations presented here become deterministic. Empirical results suggest, however, that these biases have a significant impact on finite populations as well [Poli and McPhee, 2001b].

2 Truncation

Since we are only concerned with the lengths of strings, we will model a population by recording the proportion of members it has of each length. Let $p_t(k)$ be the proportion of members with length $k = 0, 1, 2, \ldots$ for a population at time t, so that $\sum_k p_t(k) = 1$. We will assume that the initial population contains no strings of length zero, that is, $p_0(0) = 0$, and the operators which we will consider ensure that $p_t(0) = 0$ for $t > 0$.

We may take p_t to be a probability distribution over lengths. Let X_t be a random variable so distributed and let T be the random variable representing length after truncation. Note that

$$\mathbf{E}[X_t] = \sum_{j>0} j p_t(j)$$

The probability of truncation producing a prefix string of length k may be calculated by summing over the probabilities that a string of length $j > k$ is selected as the parent and that it is then truncated in the right place. The truncation cut-point is selected uniformly at random along the string's length. We allow cuts before the first element but not after the last, as the terminal node of the program must be thrown away in this parent. We then get

$$\mathbf{Pr}[T = k] = \sum_{j>k} \frac{p_t(j)}{j}$$

The expected value of T is

$$\begin{aligned}
\mathbf{E}[T] &= \sum_{k \geq 0} k \sum_{j>k} \frac{p_t(j)}{j} \\
&= \sum_{j>0} \frac{p_t(j)}{j} \sum_{k=0}^{j-1} k \\
&= \sum_{j>0} \frac{p_t(j)}{j} \left(\frac{j(j-1)}{2} \right) \\
&= \frac{1}{2} \sum_{j>0} p_t(j)(j-1) \\
&= \frac{1}{2} \left(\sum_{j>0} j p_t(j) - \sum_{j>0} p_t(j) \right) \\
&= \frac{1}{2} (\mathbf{E}[X_t] - 1)
\end{aligned}$$

Similarly, the variance is

$$\begin{aligned}
\mathbf{Var}[T] &= \mathbf{E}[T^2] - \mathbf{E}[T]^2 \\
&= \sum_{k \geq 0} k^2 \sum_{j>k} \frac{p_t(j)}{j} - \mathbf{E}[T]^2 \\
&= \sum_{j>0} \frac{p_t(j)}{j} \sum_{k=0}^{j-1} k^2 - \mathbf{E}[T]^2 \\
&= \sum_{j>0} \frac{p_t(j)}{j} \left(\frac{(j-1)j(2j-1)}{6} \right) - \mathbf{E}[T]^2 \\
&= \frac{1}{6} \sum_{j>0} p_t(j)(2j^2 - 3j + 1) - \mathbf{E}[T]^2 \\
&= \frac{1}{3} \sum_{j>0} j^2 p_t(j) - \frac{1}{2} \sum_{j>0} j p_t(j) + \frac{1}{6} \sum_{j>0} p_t(j) \\
&\quad - \frac{1}{4} (\mathbf{E}[X_t] - 1)^2 \\
&= \frac{1}{3} \mathbf{E}[X_t^2] - \frac{1}{2} \mathbf{E}[X_t] + \frac{1}{6} \\
&\quad - \frac{1}{4} \mathbf{E}[X_t]^2 + \frac{1}{2} \mathbf{E}[X_t] - \frac{1}{4}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{3}\mathbf{E}[X_t^2] - \frac{1}{3}\mathbf{E}[X_t]^2 + \frac{1}{12}\mathbf{E}[X_t]^2 - \frac{1}{12} \\
&= \frac{1}{3}\mathbf{Var}[X_t] + \frac{1}{12}(\mathbf{E}[X_t]^2 - 1)
\end{aligned}
$$

We have assumed here that $p_t(0) = 0$, as discussed earlier.

The generating function for T will also be useful:

$$
\begin{aligned}
G_T(z) &= \sum_{k\geq 0} \mathbf{Pr}[T=k]z^k \\
&= \sum_{k\geq 0}\sum_{j>k} \frac{p_t(j)}{j} z^k \\
&= \sum_{j>0} \frac{p_t(j)}{j} \sum_{k=0}^{j-1} z^k \\
&= \sum_{j>0} \frac{p_t(j)}{j} \left(\frac{1-z^j}{1-z}\right) \\
&= \frac{1}{1-z}\left(\sum_{j>0}\frac{p_t(j)}{j} - \sum_{j>0}\frac{p_t(j)}{j} z^j\right)
\end{aligned}
$$

3 Crossover

Crossover creates the string to be concatenated by selecting randomly from the population, and applying truncation to the selected string. However, there is an asymmetry, as now we throw away between 0 and $k-1$ elements from a string of length k. Thus we always preserve at least one element in the concatenation string (the terminal of the program). This justifies (for crossover) the assumption that $p_t(0) = 0$, for $t > 0$.

Let C be the length of strings after applying this truncation to a copy of the population. That is, T represents the length of the contribution of one parent (the left-hand side, or prefix) and C represents the length of the contribution of the other parent (the right-hand side, or suffix). Then

$$
\mathbf{Pr}[C=k] = \sum_{j\geq k}\frac{p_t(j)}{j}
$$

for $k > 0$, with $\mathbf{Pr}[C = 0] = 0$, indicating that we always add on at least one element. The probability distribution for string lengths at time $t+1$ (resulting from adding T and C) is then

$$
\begin{aligned}
p_{t+1}(k) &= \sum_n \mathbf{Pr}[T=n]\mathbf{Pr}[C=k-n] \\
&= \sum_{n=0}^{k-1}\sum_{i>n}\frac{p_t(i)}{i}\sum_{j\geq k-n}\frac{p_t(j)}{j}
\end{aligned}
$$

(see [McPhee et al., 2001] for an alternative form of this equation).

Following arguments similar to those above for T, we have

$$
\mathbf{E}[C] = \frac{1}{2}(\mathbf{E}[X_t]+1)
$$

and

$$
\mathbf{Var}[C] = \frac{1}{3}\mathbf{Var}[X_t] + \frac{1}{12}(\mathbf{E}[X_t]^2 - 1) = \mathbf{Var}[T]
$$

If we let X_{t+1} denote the string length in a population at time $t+1$, we can easily calculate the mean and variance of this random variable following the application of crossover, since

$$
\mathbf{E}[X_{t+1}] = \mathbf{E}[T] + \mathbf{E}[C]
$$

and

$$
\mathbf{Var}[X_{t+1}] = \mathbf{Var}[T] + \mathbf{Var}[C] = 2\mathbf{Var}[T]
$$

Thus

$$
\begin{aligned}
\mathbf{E}[X_{t+1}] &= \frac{1}{2}(\mathbf{E}[X_t]-1) + \frac{1}{2}(\mathbf{E}[X_t]+1) \\
&= \mathbf{E}[X_t]
\end{aligned}
$$

and

$$
\begin{aligned}
\mathbf{Var}[X_{t+1}] &= 2\mathbf{Var}[T] \\
&= \frac{2}{3}\mathbf{Var}[X_t] + \frac{1}{6}(\mathbf{E}[X_t]^2 - 1)
\end{aligned}
$$

That is, the mean length is not changed by crossover, although the variance is [McPhee et al., 2001]. We can get an idea of how much the variance changes by seeing what happens if we repeatedly apply crossover to a population. Since the mean length remains fixed, we can solve the recurrence for $\mathbf{Var}[X_t]$ giving

$$
\begin{aligned}
\mathbf{Var}[X_t] &= \left(\frac{2}{3}\right)^t \left(\mathbf{Var}[X_0] - \frac{1}{2}(\mathbf{E}[X_0]^2 - 1)\right) \\
&\quad + \frac{1}{2}(\mathbf{E}[X_0]^2 - 1)
\end{aligned}
$$

We have here made an infinite population assumption so that we can ignore all stochastic fluctuations from one generation to the next which would occur with a finite population. Example curves are plotted in Figure 1. This result tells us that the variance move exponentially quickly towards a fixed point $(\mathbf{E}[X_0]^2 - 1)/2$, and that, therefore, crossover has a strong bias towards this variance. In fact, fixed point distributions for repeated crossover has been calculated [McPhee et al., 2001]:

Theorem 1 *Distributions of the form*

$$p(k) = (1-a)^2 k a^{k-1}$$

are fixed-points for repeated crossover, where $0 < a < 1$ *is a parameter corresponding to a mean value of* $p(k)$ *of*

$$\mu = \frac{1+a}{1-a}$$

Proof

Assume we have the given distribution. Then the next distribution is:

$$\begin{aligned}
& p_{t+1}(k) \\
= & \sum_{n=0}^{k-1} \sum_{i>n} \frac{p(i)}{i} \sum_{j \geq k-n} \frac{p(j)}{j} \\
= & \sum_{n=0}^{k-1} \sum_{i>n} (1-a)^2 a^{i-1} \sum_{j \geq k-n} (1-a)^2 a^{j-1} \\
= & (1-a)^4 \sum_{n=0}^{k-1} \sum_{i>n} a^{i-1} \left(\frac{a^{k-n}}{a(1-a)} \right) \\
= & \frac{(1-a)^3}{a} \sum_{n=0}^{k-1} a^{k-n} \sum_{i>n} a^{i-1} \\
= & \frac{(1-a)^3}{a} \sum_{n=0}^{k-1} a^{k-n} \left(\frac{a^{n+1}}{a(1-a)} \right) \\
= & \frac{(1-a)^2}{a^2} \sum_{n=0}^{k-1} a^{k+1} \\
= & \frac{(1-a)^2}{a^2} k a^{k+1} \\
= & (1-a)^2 k a^{k-1}
\end{aligned}$$

□

Examples of this distribution for different values of the mean are plotted in Figure 2. The exponential convergence of the variance indicates that the application of crossover is strongly biased towards distributions of this kind. Experimental results corroborating this conclusion may be found in [McPhee et al., 2001].

4 Full mutation

With full mutation, the string to be added is always of a fixed length $d > 0$. This again justifies the assumption that $p_t(0) = 0$. It is easy to see then that with this operator $\mathbf{E}[C] = d$ and $\mathbf{Var}[C] = 0$. Then we have

$$\mathbf{E}[X_{t+1}] = \frac{1}{2}(\mathbf{E}[X_t] - 1) + d$$

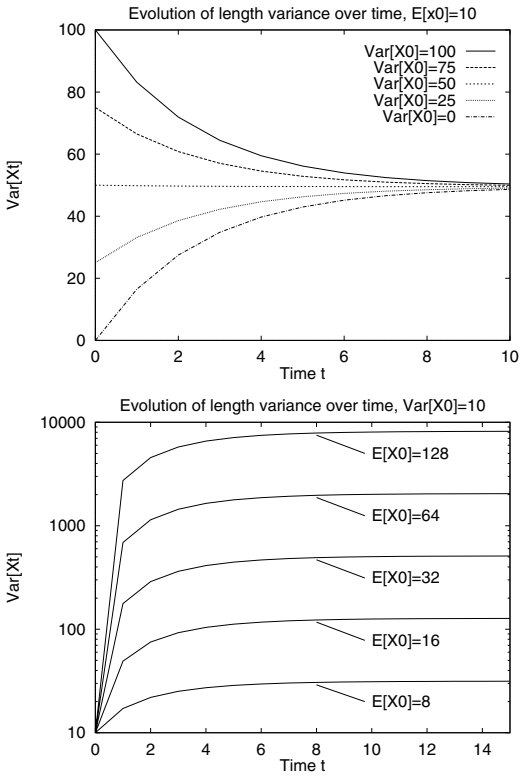


Figure 1: The variance of the lengths of strings in an infinite population with repeated application of crossover, for different starting distributions. In the top picture, the mean of the initial distribution is fixed at 10 for all plots, and the initial variance is altered. In the bottom picture, the initial variance is fixed and plots are shown for different initial mean lengths (on a logarithmic scale). The application of crossover does not affect the mean length from one generation to the next.

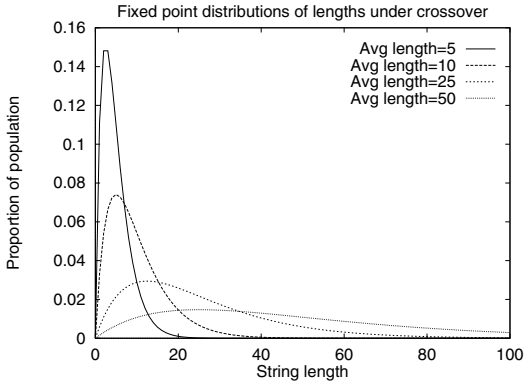


Figure 2: The fixed-point distribution for repeated application of crossover, for various mean lengths. The application of crossover does not affect the mean length.

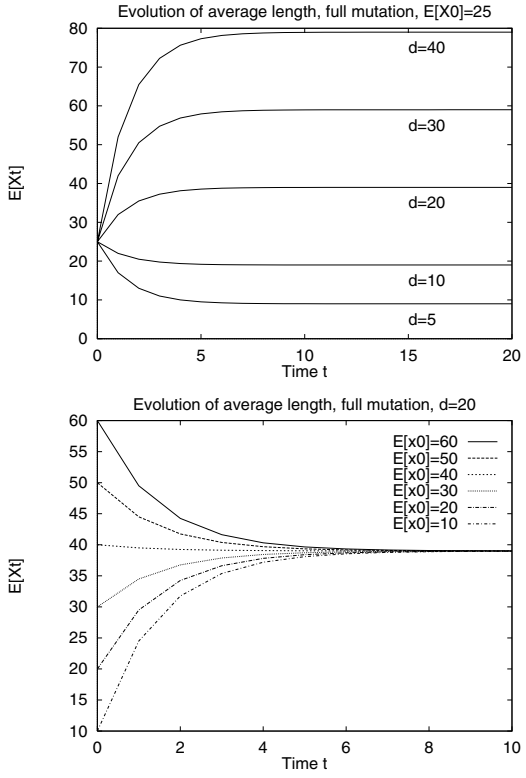


Figure 3: The mean length of an infinite population with repeated application of full mutation, for various initial distributions and values of d.

and

$$\mathbf{Var}[X_{t+1}] = \frac{1}{3}\mathbf{Var}[X_t] + \frac{1}{12}(\mathbf{E}[X_t]^2 - 1)$$

Thus both the mean and variance change over time. Again, considering the repeated application of this operator (in the infinite population limit), we find the mean converges to a limit exponentially:

$$\mathbf{E}[X_t] = \left(\frac{1}{2}\right)^t (\mathbf{E}[X_0] - 2d + 1) + 2d - 1$$

Example curves are plotted in Figure 3. Thus full mutation biases the mean length of strings towards the fixed-point

$$\lim_{t\to\infty} \mathbf{E}[X_t] = 2d - 1$$

The equation for the variance is harder to solve, since it depends on the mean. However, a recurrence for the fixed-point distribution can be found using generating functions (see [Graham et al., 1994] for an introduction to this technique). The generating function for the truncated (prefix) string, calculated above, is

$$G_T(z) = \frac{1}{1-z}\left(\sum_{j>0}\frac{p(j)}{j} - \sum_{j>0}\frac{p(j)}{j}z^j\right)$$

For full mutation

$$G_C(z) = \sum_{k\geq 0}\mathbf{Pr}[C=k]z^k = z^d$$

Adding d to the result of truncation therefore gives us a distribution with generating function $z^d G_T(z)$. Let the generating function for the fixed-point distribution be

$$\begin{aligned} G_X(z) &= G_{T+C}(z) \\ &= G_T(z)G_C(z) \\ &= \frac{z^d}{1-z}\left(\sum_{j>0}\frac{p(j)}{j} - \sum_{j>0}\frac{p(j)}{j}z^j\right) \end{aligned}$$

which means

$$(1-z)\sum_{k\geq 0}p(k)z^k = z^d\left(\sum_{j>0}\frac{p(j)}{j}\right) - \sum_{j>0}\frac{p(j)}{j}z^{j+d}$$

We now compare coefficients of z^k from both sides of this equation. Looking at the coefficient of z^0 tells us that $p(0) = 0$ as expected. For all $0 < k < d$ we find

$$p(k) - p(k-1) = 0$$

which tells us that $p(k) = 0$ for these cases. However, for $k = d$ we get

$$p(d) - p(d-1) = \sum_{j>0}\frac{p(j)}{j}$$

and therefore

$$p(d) = \sum_{j>0}\frac{p(j)}{j}$$

since $p(d-1) = 0$. For $d < k < 2d$ we have

$$p(k) - p(k-1) = -\frac{p(k-d)}{k-d} = 0$$

therefore

$$p(k) = p(k-1) = \sum_{j>0}\frac{p(j)}{j}$$

for $d < k < 2d$. Finally for $k > 2d$ we get

$$p(k) - p(k-1) = -\frac{p(k-d)}{k-d}$$

which gives us a recurrence relation

$$p(k) = p(k-1) - \frac{p(k-d)}{k-d}$$

To summarise:

$$p(k) = \begin{cases} 0 & \text{if } k < d \\ \sum_{j>0} \frac{p(j)}{j} & \text{if } d \leq k < 2d \\ p(k-1) - \frac{p(k-d)}{k-d} & \text{if } k \geq 2d \end{cases}$$

We see in general that the distribution is zero for $k < d$ and a constant value for $d \leq k < 2d$, after which it tails off according to the given recurrence (see [McPhee et al., 2001] for experimental evidence of this). For $d = 1$ one can verify that the solution becomes

$$p(k) = [k = 1]$$

that is, a single spike at $k = 1$. For $d = 2$ we have the following:

Theorem 2 *The fixed-point distribution for repeated application of full mutation with $d = 2$ is*

$$p(k) = \frac{[k > 1]}{e(k-2)!}$$

That is, the distribution is zero for $k = 0, 1$, it takes the constant value e^{-1} for $k = 2, 3$ after which it tails off following an inverse factorial.

Proof

The cases $k = 0, 1$ are obvious. To prove the remainder, we will first show that

$$p(k) = \frac{\alpha}{(k-2)!}$$

for some constant α, when $k > 1$. Define

$$\alpha = \sum_{j>0} \frac{p(j)}{j}$$

Now proceed by induction. When $k = 2, 3$ we have

$$p(k) = \alpha$$

which is correct. Now assume we are correct for all values below some $k > 3$. Then

$$\begin{aligned} p(k) &= p(k-1) - \frac{p(k-2)}{k-2} \\ &= \frac{\alpha}{(k-3)!} - \frac{\alpha}{(k-2)(k-4)!} \\ &= \frac{(k-2)\alpha}{(k-2)!} - \frac{(k-3)\alpha}{(k-2)!} \\ &= \frac{\alpha}{(k-2)!} \end{aligned}$$

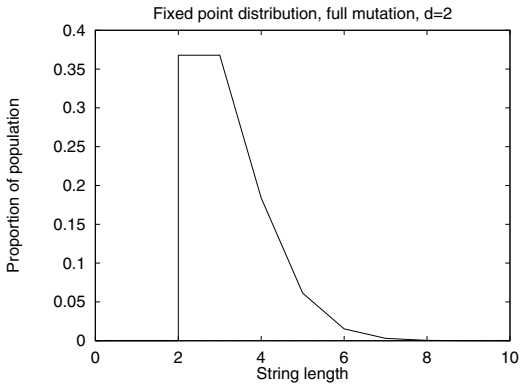


Figure 4: The fixed-point distribution of lengths for repeated application of full mutation with $d = 2$.

The fact that

$$\sum_{k \geq 0} p(k) = 1$$

means that we find $\alpha = e^{-1}$, since

$$\begin{aligned} \sum_{k \geq 0} p(k) &= \sum_{k \geq 2} \frac{\alpha}{(k-2)!} \\ &= \alpha \sum_{k \geq 0} \frac{1}{k!} \\ &= \alpha e \end{aligned}$$

□

Figure 4 shows this distribution for $d = 2$.

5 Growth mutation

Growth mutation is where the string to be concatenated is grown according to a geometric distribution, where there is a probability q that an extra element will be added to the string. The probability that a string of length k will be added is therefore $q^{k-1}(1-q)$. In this case

$$\mathbf{E}[C] = \frac{1}{1-q}$$

and

$$\mathbf{Var}[C] = \frac{q}{(1-q)^2}$$

so the mean length of the population of final strings is

$$\mathbf{E}[X_{t+1}] = \frac{1}{2}(\mathbf{E}[X_t] - 1) + \frac{1}{1-q}$$

Similarly, the variance is

$$\mathbf{Var}[X_{t+1}] = \frac{1}{3}\mathbf{Var}[X_t] + \frac{1}{12}(\mathbf{E}[X_t]^2 - 1) + \frac{q}{(1-q)^2}$$

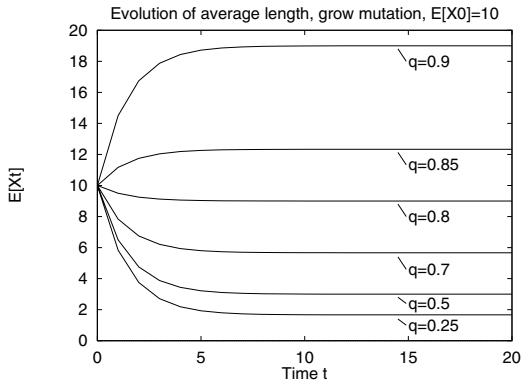


Figure 5: The mean length of an infinite population with repeated application of growth mutation, for various values of q.

Iterating the equation for the mean length gives us

$$\mathbf{E}[X_t] = \left(\frac{1}{2}\right)^t \left(\mathbf{E}[X_0] - \frac{1+q}{1-q}\right) + \frac{1+q}{1-q}$$

which indicates that the average length is biased towards $(1+q)/(1-q)$. Example curves are plotted in Figure 5.

Again, the equation for the variance is hard to solve, but we can find an expression for the fixed-point distribution. Remarkably, it is identical in form to that given for crossover.

Theorem 3 *The fixed-point distribution for repeated application of growth mutation (with growth probability q) is*

$$p(k) = (1-q)^2 k q^{k-1}$$

Proof

Let $G_C(z)$ be the generating function for the concatenating string length:

$$\begin{aligned} G_C(z) &= \sum_{k>0} q^{k-1}(1-q)z^k \\ &= \sum_{k>0} q^{k-1}z^k - \sum_{k>0} q^k z^k \\ &= z\sum_{k\geq 0} q^k z^k - \left(\sum_{k\geq 0} q^k z^k - 1\right) \\ &= (z-1)\sum_{k\geq 0} q^k z^k + 1 \\ &= \frac{z-1}{1-qz} + 1 \\ &= \frac{z-qz}{1-qz} \end{aligned}$$

Then the generating function for the result of growth mutation is

$$G_X(z) = G_{T+C}(z) = G_T(z)G_C(Z)$$

where $G_T(z)$ is the generating function for the truncated string lengths defined previously. That is:

$$G_X(z) = \frac{z(1-q)}{(1-z)(1-qz)}\left(\sum_{j>0}\frac{p(j)}{j} - \sum_{j>0}\frac{p(j)}{j}z^j\right)$$

So at the fixed-point:

$$\begin{aligned} &(1-z)(1-qz)\sum_{k\geq 0} p(k)z^k \\ = \;& z(1-q)\left(\sum_{j>0}\frac{p(j)}{j} - \sum_{j>0}\frac{p(j)}{j}z^j\right) \end{aligned}$$

Again, comparing coefficients on either side of the equation gives us

$$\begin{aligned} p(0) &= 0 \\ p(1) &= (1-q)\sum_{j>0}\frac{p(j)}{j} \end{aligned}$$

and

$$p(k) = \left(1 + q - \frac{1-q}{k-1}\right)p(k-1) - qp(k-2)$$

for $k > 1$. Setting

$$\alpha = \sum_{j>0}\frac{p(j)}{j}$$

we first prove by induction that

$$p(k) = \alpha(1-q)kq^{k-1}$$

This is correct for $k = 0, 1$. Now assume it is correct below some value $k > 1$. Then

$$\begin{aligned} p(k) &= \left(1 + q - \frac{1-q}{k-1}\right)p(k-1) - qp(k-2) \\ &= \left(1 + q - \frac{1-q}{k-1}\right)\alpha(1-q)(k-1)q^{k-2} \\ &\quad -q\alpha(1-q)(k-2)q^{k-3} \\ &= \alpha(1-q)q^{k-2}((1+q)(k-1) - (1-q) \\ &\quad -(k-2)) \\ &= \alpha(1-q)kq^{k-1} \end{aligned}$$

as required. The fact that

$$\sum_{k\geq 0}\alpha(1-q)kq^{k-1} = 1$$

allows us to deduce that $\alpha = (1 - q)$ and the result follows.

□

Since the fixed-point for growth mutation is also a fixed-point for crossover, it must also be a fixed-point for the application of both operators applied sequentially.

6 Discussion

In [Poli and McPhee, 2001b, McPhee et al., 2001, Poli and McPhee, 2001a] GP schema theory is used to analyze the size biases induced by crossover and mutation when using linear representations and flat fitness landscapes. This paper extends several of those results, and provides proofs for others. While we have closed forms for many of the quantities explored here, we do not yet have closed forms for others (e.g., the variances for the two mutation operators); filling these gaps would be an obvious extension of this work. Another extension would be to try to apply the techniques used here to the study of more complex systems (e.g., non-flat fitnesses or non-linear tree structures).

Taken as a group, the results presented here help build at least the beginnings of a picture of the size biases of some of the most commonly used operators in GP. In all three cases, for example, the average length of strings either remains constant or quickly approaches some limit. This implies, for example, that none of these operators induce unbounded bloat on a flat fitness landscape; in fact something like the reverse is true in the sense that all three operators heavily oversample shorter strings (see [Poli and McPhee, 2001b, McPhee et al., 2001, McPhee and Poli, 2001] for details). These results also make it clear that the average and the variance of the length of the (infinite) populations move towards their limit values very quickly, often reaching near convergence in less than 10 generations. This suggests that these operators induce quite strong biases, and these biases may have an impact even in problems with non-flat fitness landscapes.

Acknowledgement

The second author would like to extend special thanks to The University of Birmingham School of Computer Science for graciously hosting him during his sabbatical, and various offices and individuals at the University of Minnesota, Morris, for making that sabbatical possible.

References

[Graham et al., 1994] Graham, R. L., Knuth, D. E., and Patashnik, O. (1994). *Concrete Mathematics*. Addison-Wesley, second edition.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA.

[McPhee and Poli, 2001] McPhee, N. F. and Poli, R. (2001). A schema theory analysis of the evolution of size in genetic programming with linear representations. In *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, Milan. Springer-Verlag.

[McPhee et al., 2001] McPhee, N. F., Poli, R., and Rowe, J. E. (2001). A schema theory analysis of mutation size biases in genetic programming with linear representations. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul, Korea.

[Poli and McPhee, 2001a] Poli, R. and McPhee, N. F. (2001a). Exact GP schema theory for headless chicken crossover and subtree mutation. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul, Korea.

[Poli and McPhee, 2001b] Poli, R. and McPhee, N. F. (2001b). Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, Milan. Springer-Verlag.

The Mixture of Trees Factorized Distribution Algorithm

Roberto Santana, Alberto Ochoa-Rodriguez, Marta R. Soto
Center of Mathematics and Theoretical Physics.
ICIMAF. Calle 15, e/ C y D, Vedado CP 10400. C-Habana. Cuba
{rsantana,ochoa,mrosa}@cidet.icmf.inf.cu

Abstract

This paper introduces a Factorized Distribution Algorithm based on a mixture of trees distribution. The probabilistic model and the learning algorithm used differs to previous uses of probabilistic modeling in the context of Evolutionary Computation. Preliminary results show the algorithm is competitive, and some times superior to other Factorized Distribution Algorithms. We also illustrate how particular features of the search space can be employed during the search by conveniently selecting the mixture of trees parameters.

1 INTRODUCTION

Factorized Distribution Algorithms (FDAs) (Mühlenbein, Mahnig, & Ochoa, 1999) are population based search methods that combine results from Graphical Models and Evolutionary Computation research, and are considered as a tractable subclass of Estimation Distribution Algorithms (Mühlenbein & Paaß, 1996). In order to optimize a given function they begin by generating an initial random population of points which are evaluated using the objective function. Some of the points are selected based on their values, and a factorized probabilistic model of their underlying distribution is constructed. This probabilistic model is used to sample the points that will be part of the next population.

A number of FDAs that use probabilistic models based on dependency trees have shown up in the literature. In (Baluja & Davies, 1997) a tree factorization computed using the Chow and Liu algorithm (Chow & Liu, 1968) is employed to approximate the underlying distribution of the selected points. The Bivariate Marginal Distribution Algorithm (BMDA) (Pelikan & Mühlenbein, 1999) uses a forest instead of a tree based factorization. Only second order statistics are computed.

The Polytree Approximation Distribution Algorithm (PADA) has been designed to deal with the class of single connected Bayesian Networks (BNs) (Soto *et al.*, 1999). A revised version of this algorithm has been presented in (Ochoa, Muehlenbein, & Soto, 2000) where the previous introduced algorithms based on trees and forest distributions are covered.

Modeling by finite mixture of distributions (Everitt & Hand, 1981) concerns modeling a statistical distribution by a mixture (or weighted sum) of other distributions. Recently, the research on mixture models has begun to receive a particular attention by the EDAs community. In (Thierens & Bosman, 2001) the authors use a mixture of Gaussian probabilistic density functions for the solution of continuous multi-objective functions. In (Peña, Lozano, & Larrañaga, 2001) authors employ mixture models as the basis for data clustering in multimodal continuous and discrete function optimization via EDAs. In the case of discrete optimization a framework for learning mixture of BNs that share the same structure is presented.

Mixture of distributions have been used also in the framework of evolutionary optimization that use flexible probability estimators (Gallagher, Frean, & Downs, 1999) (an approach very close to FDAs) where the adaptive mixture model of Priebe is used to estimate probability distributions in an evolutionary optimization context.

2 TREES AND MIXTURE OF TREES MODELS

Mixtures of trees belong to the class of finite mixture distributions. They were introduced in (Meila, 1999) and are the core of the optimization algorithm we present in this paper. Now the probabilistic models are formally introduced. We will utilize the same notation used in (Meila, 1999).

Let V denote the set of variables of our problem. According to the graphical model paradigm, each variable is viewed as a vertex of an (undirected) graph $G = (V, E)$ which is called a tree if it has no cycles. Now we define a probability distribution T that is conformal with a tree.

$$T(x) = \prod_{v \in V} T_{v|pa(v)}(x_v|x_{pa(v)}) \tag{1}$$

The distribution T itself will be called a tree when no confusion is possible. The graph (V, E) represents the structure of the distribution T.

A mixture of trees is defined to be a distribution of the form:

$$Q(x) = \sum_{k=1}^{m} \lambda_k T^k(x) \tag{2}$$

with $\lambda_k \geq 0$, $k = 1, .., m$, $\sum_{k=1}^{m} \lambda_k = 1$.

The tree distributions are the mixture components, and the λ_k are called mixture coefficients. A mixture of trees can be viewed as containing an unobserved choice variable z, which takes values $k \in \{1, ..., m\}$ with probability λ_k. Conditioned on the value of z the distribution of the visible variables V is a tree. The m trees may have different structures and different parameters.

3 THE LEARNING ALGORITHM IN THE CONTEXT OF THE MT-FDA

We present first an algorithm for fitting a mixture of trees to an observed data set in the Maximum Likelihood paradigm via the Expectation-Maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977). The mixtures of trees learning algorithm constitutes a building block for the FDA presented here, this algorithm can be found in (Meila, 1999) where it was introduced. We resume some of its features:

The learning problem is: Given a set of observations $D = \{x^1, x^2, , x^N\}$, we are required to find the mixture of trees Q that satisfies

$$Q = \arg\max_{Q} \sum_{i=1}^{N} \log Q(x^i) \tag{3}$$

The EM algorithm introduces a likelihood function called the *complete log − likelihood* which is the log-likelihood of both, the observed and the unobserved data, given the current model estimate $M = \{m, T^k, \lambda_k, k = 1, ...m\}$

$$lc(x^{1...N}, z^{1...N}|M) = \sum_{i=1}^{N} \sum_{k=1}^{m} \delta_{k,z^i} (\log \lambda_k + \log T^k(x^i)) \tag{4}$$

where δ_{k,z^i} is equal to one if z^i is equal to the kth value of the choice variable, and zero otherwise.

The idea underlying the EM algorithm is to compute and optimize the expected value of lc. In the context of population based evolutionary optimization our objective is to find a factorization of the probability distribution of the selected set of points using a mixture of trees. We have called our algorithm Mixture of Trees FDA (MT-FDA).

In figure 1 the pseudo-code of the MT-FDA is presented. The first population is randomly generated by the algorithm. From the current population, a subset of points is selected. A mixture of trees Q that fits the selected set is found using the mixture of trees learning algorithm that takes as parameters the number of trees, and a schedule with the number of learning steps in each generation. New points are generated by sampling from Q. The best elitism scheme is used, where all the individuals selected in the current population pass to the next one. Different replacing strategies can be used to combine points from the current population and new generated points in the next population.

Mixture of trees FDA (MT − FDA)

- **STEP 0:** Set $t \Leftarrow 0$. Generate $N \gg 0$ points randomly.
- **STEP 1:** Select a set S of $k < N$ points according to a selection method.
- **STEP 2:** Calculate a mixture of trees Q that approximates S using the mixture of trees learning algorithm.

- **STEP 3:** Generate $N - k$ new points sampling from Q.
- **STEP 4:** Combine in the new population the k selected points with the $N - k$ new points. Set $t \Leftarrow t + 1$
- **STEP 5:** If the termination criteria are not met, go to STEP 1

Figure 1: MT-FDA

The determination of the extent of learning to be allowed is a sensitive issue for the MT-FDA. When sampling from the learned model, the two traditional goals of an efficient search, exploitation and exploration, have to be accomplished. A model that best approximates the data can less likely generate, during the sampling step, points that belong to unexplored areas of the search space. So, it is advisable to stop the learning algorithm before the improvement in the likelihood ceases. The number of trees also influence the likelihood. The design of strategies that set the appropriate schedule for the learning steps, and of criteria to determine the number of trees for a given optimization problem, are topics where further research is required.

We test two methods for selecting a convenient starting mixture of trees. Both methods begin by initializing each component of the mixture using the Chow and Liu's algorithm. They differ in the second step. The first method makes perturbations to the structure of each tree. The parent of one of the leaf nodes is replaced by another randomly selected node. By making this change on a leaf node the procedures guarantees no cycle will be formed in the tree after the perturbation.

The second method keeps intact the tree structure and makes a perturbation to the initial probability values $P^k(x^i)$, in order to let the learning algorithm change the structure when fitting the perturbed probabilities. Recall that, independently of the selected root, Chow and Liu's algorithm guarantees that all trees are equivalent in their representation of the data. By applying a small perturbation on $P^k(x^i)$ values, identicalness among the trees is broken. Both methods guarantee solutions superior to random initialization.

Mixture distributions have a number of distinctive attributes that make them particularly appealing for their use in the framework of FDAs. Maybe the most important is the possibility of representing, condensed in just one model, different patterns of interactions among the variables of the problem. In Bayesian Networks the change in one variable's value can determine changes only in the parameters of other variables, not in their structural relation. In mixture of finite distributions the structure of dependencies among a set of variables can change depending on the values of the choice variable they depend on. This fact can be used in a flexible way for incorporating diverse search strategies into the MT-FDA.

Now we present some alternative ways the choice variable can be incorporated by an FDA whose model is a mixture of trees. The analysis is divided in two scenarios, when the choice variable is hidden, and when it is known.

3.1 The choice variable z is hidden.

Two cases can be distinguished in this scenario. The first is when, although the choice variable is unknown, it is in fact one (or a small subset) of the variables of the problem. In this case a fitter approximation of the selected points by means of a mixture of trees could be found by identifying the set of variables. There exist methods for identifying these variables from the analysis of data (Meila, 1999). Although we do not explore this trend in the present work, we hypothesize that this identification task can be inserted in the evolutionary process.

The second case, and the one extensively considered in our experiments, is when there is no previous information about the choice variable, but we try to fit the target distribution with a mixture that has m trees, based on an unknown variable that could help to cluster the space of solutions. As the existence of a mixture of trees based on such a kind of choice variable is an assumption, there could be cases where the mixture of trees does not lead to an appropriate factorization.

3.2 The choice variable z is known

Again we analyze two particular cases: When the choice variable is in fact a variable, (or the subset of the variables) of the problem. And, when the choice variable z does not belong to the set of variables of the problem but we can decide its values based on a predefined criterium. We mention two cases:

a) z is related to a genotype clustering of the set of points

b) z is related to a phenotype clustering of the set of points.

It has been shown (Pelikan & Goldberg, 2000) that for certain type of problems a genotype clustering of the population can contribute to an efficient search. The mixture of trees model can be used not only to cluster the space of solutions but also to exchange informa-

tion among the clusters during the evolution. When solutions could be classified in regions based on their structural similarities, and a mapping between solutions that belong to the same class and their fitness evaluation exists, it is expected that a mixture of trees based on a choice variable describing the classes could lead to a good approximation.

4 EXPERIMENTS

The experiments were designed to illustrate the behavior of the proposed algorithm and compare its performance with other FDAs. First we introduce the functions that were employed. Then we present a comparison between the MT-FDA and a tree based FDA. Some experiments are presented to illustrate the difference in the performance of the Bayesian FDAs and the MT-FDA. Finally an example is shown on the convenience of using one of the variables of the problem as the choice variable.

Let u be the number of bits turned on in the string x. A function of unitation is a function whose value depends only on the number of ones on an input string. The values of the strings with the same number of ones are equal. Deceptive functions are defined as a sum of more elementary deceptive functions f_k of k variables.

$$f(x) = \sum_{j=1}^{l} f_k(s_j), \tag{5}$$

where s_j are non-overlapping substrings of x containing k elements.

Function OneMax:

$$OneMax(x) = \sum_{i=1}^{n} x_i \tag{6}$$

Function f^3_{dec}:

$$f^3_{dec} = \begin{cases} 0.9 & for \quad u = 0 \\ 0.8 & for \quad u = 1 \\ 0.0 & for \quad u = 2 \\ 1.0 & for \quad u = 3 \end{cases} \tag{7}$$

Function $f_{3deceptive}$:

$$f_{3deceptive}(X) = \sum_{i=1}^{i=\frac{n}{3}} f^3_{dec}(X_{3i-2}, X_{3i-1}, X_{3i}) \tag{8}$$

Function $Isotorus$:

u	0	1	2	3	4	5
$IsoT_1$	m	0	0	0	0	$m-1$
$IsoT_2$	0	0	0	0	0	m^2

(9)

$$F_{IsoTorus} = \tag{10}$$

$$\sum_{i=2}^{n} IsoT_2(x_{up}, x_{left}, x_i, x_{right}, x_{down}) \tag{11}$$

$$+ \quad IsoT_1(x_{1-m+n}, x_{1-m+n}, x_1, x_2, x_{1+m}) \tag{12}$$

where x_{up}, etc., are defined as the appropriate neighbors, wrapping around.

Function $BigJump$:

$$BigJump = \begin{cases} u & for \quad 0 \leq u \leq n-m \\ 0 & for \quad n-m \leq u \leq m \\ k \cdot n & for \quad u = m \end{cases} \tag{13}$$

Function Nf^3_{dec}:

$$Nf^3_{dec} = \begin{cases} \sum_{i=1}^{i=\frac{n-1}{3}} f^3_{dec}(X_{3i-1}, X_{3i}, X_{3i}+1), \; if \; x_1 = 1 \\ \sum_{i=1}^{i=\frac{n-1}{3}} (1 - f^3_{dec}(X_{3i-1}, X_{3i}, X_{3i}+1), \; oth. \end{cases} , \tag{14}$$

4.1 Comparison between a tree based FDA and the MT-FDA

First we make a comparison between a tree based FDA and the MT-FDA. The tree based FDA has the same pseudo-code as the MT-FDA presented in section 4, but only one tree found using the Chow and Liu's algorithm is employed. The main difference between Baluja's algorithm and our tree based FDA is that we do not update the bivariate probabilities multiplying by a decay factor, instead in every generation, bivariate probabilities are calculated from the selected set. We study the difference in the behavior of the tree based FDA and the MT-FDA for diverse truncation values and functions.

Common parameters for the experiments were: Population size = 1000, Best elitism, Maxgen =20. Stopping criteria were: the maximum number of generations, and a total homogeneity in the selected population (i.e. all the individuals were identical).

Function	n	τ	Tree		MT-FDA	
			S	Gen.	S	Gen.
	30	0.05	37	4.92	9	6.33
$f_{3deceptive}$	30	0.1	60	5.73	63	6.66
	30	0.15	72	6.84	75	7.8
	30	0.2	65	8.28	75	9.28
	30	0.25	85	8.8	95	9.56
$Isotorus$	36	0.05	80	3.49	75	4.05
	36	0.1	84	4.4	93	4.9
	36	0.15	85	5.19	92	5.78
	36	0.2	84	6.14	90	6.5
	36	0.25	71	6.86	91	7.38

Table 1: Numerical results for MT-FDA and tree based FDA

The schedule for learning was the following: In the first population there were 10 learning steps to learn the mixture, in the following generations 5 learning steps. Recall that for the MT-FDA the trees from which the learning algorithm is started are also found using the Chow and Liu's algorithm. So this initialization can be seen as the best we could achieve with just one tree (what the tree based FDA actually does), but in the following learning steps the likelihood of data given by the initial trees is improved.

In Table 1 are presented results for two different functions. n is the number of variables, τ the truncation parameter, S the number of times the optimum was found in 100 runs and $Gen.$ the average number of generations needed to find the optimum. Initial experiments were made for $f_{3deceptive}$ with $\tau = 0.05$. Results were not particularly heart warming. In the table it can be appreciated that for this truncation value the tree based FDA finds the optimum four more times than when MT-FDA is used. However, for, $\tau = 0.25$ the MT-FDA clearly surpasses the tree with an impressive 95 percent of success. Similar results were achieved for the Isotorus function.

4.2 Comparison between Bayesian FDAs and the MT-FDA

In the general class of BNs the conditional probability of a variable x can depend on a subset of variables and not on only one like in the subclass of trees. All these variables are called parents. The complexity of the network is related to the maximum number of parents any variable x can have. BNs learning algorithms allow to incorporate constraints related with a maximum number of parents or the network complexity.

In (Etxeberria & Larrañaga, 1999) BNs were introduced in the framework of Evolutionary Optimization to learn, in every generation, a factorization of the selected points. In this paper we compare our algorithm with other two Bayesian FDAs. The Bayesian Optimization Algorithm (BOA) introduced in (Pelikan, Goldberg, & Cantú-Paz, 1999) and the Learning Factorized Distribution Algorithm (LFDA) (Mühlenbein & Mahnig, 1999b). Both algorithms use a Bayesian metric to measure the goodness of every Bayesian structure found, and a search procedure to search in the space of possible structures. In the BOA it is possible to set the maximum number of parents in the learnt network. For the LFDA we will refer to results appeared in (Mühlenbein & Mahnig, 1999a). Contrary to the BOA, the LFDA uses the BIC score to find the Bayesian structure and it uses a parameter α that allows to control the network complexity.

Function	n	FDA	α/ Trees	S
$OneMax$	30	LFDA	0.75	80
	30	LFDA	0.5	38
	30	LFDA	0.25	2
	30	Tree		89
	30	MT-FDA	2	77
	30	MT-FDA	5	19
	30	MT-FDA	10	0
$BigJump(30,3,1)$	30	LFDA	0.75	100
	30	LFDA	0.5	96
	30	LFDA	0.25	58
	30	Tree		99
	30	MT-FDA	2	97
	30	MT-FDA	4	91
	30	MT-FDA	6	77

Table 2: Numerical results for MT-FDA and the LFDA

A number of experiments were conducted to compare the behavior of the MT-FDA with the LFDA. In this case we used an elitism parameter of 1 (i.e. only the best individual is passed to the next population), truncation selection and Maxgen =20. Stopping criteria for MT-FDA were the same that in the previous experiments. In Table 2 column 4 refers to the number of trees and the network density for the MT-FDA and the LFDA respectively. Later, it is explained how these values were chosen.

A number of interesting features can be noticed from the analysis of the results shown in Table 2. For these functions the number of trees in the MT-FDA and the density of the BN in the LFDA play a similar role. This analogy is evident for function $BigJump$, when the number of trees or the density of the BN are in-

creased results are poorer. In the table, values used for the number of trees were selected trying to emulate the values for the density of the BN used in experiments published in (Mühlenbein & Mahnig, 2000), but we do not claim we have attained a perfect correspondence between the number of trees shown in Table 2 and the α values shown for the same functions. The analogy is related only to the progression of values for both parameters.

Results for function $Bigjump$ are an example too that when structural learning is done, simpler models can be better than more complex ones. Regarding differences between the MT-FDA and the LFDA, and beyond the analogy between the number of trees and the network density, results were very similar.

It has been acknowledged that Bayesian FDAs are very sensitive to the selection method used (Pelikan, Goldberg, & Sastry, 2000). Thus, when we claim here that one algorithm is superior to the other for the analyzed functions, the assertion is only valid for the particular selection strategy employed. We have specified the selection methods and parameters used in every comparison between Bayesian FDAs and the MT-FDA.

4.3 Comparison between BOA and the MT-FDA

We also compare our algorithm with the Bayesian Optimization Algorithm (BOA) introduced in (Pelikan, Goldberg, & Cantú-Paz, 1999). BOA uses a Bayesian metric to measure the goodness of every Bayesian structure found, and a search procedure to search in the space of possible structures. The implementation of BOA used in this paper rests on the BOA platform available at (Pelikan, Goldberg, & Sastry, 2000). It employs the Bayesian Dirichlet equivalent (BDe) metric, a greedy search procedure, and decision graphs for making the search more efficient.

Table 3 shows the experimental results of the comparison between the BOA and the MT-FDA for 2 functions. The parameters of the algorithm were: Population size = 600, Tournament selection with tournament size = 4, the worst 85 percent of the current population is replaced by the new generated individuals. The BOA was set to stop when the optimum was found, or an early convergence occurred (univariate marginals higher than 0.95). The stop criteria for MT-FDA were the same as in previous experiments, but note that in general this setting is very different from the one used in the previous experiments.

In the table the fourth column represents the maximum number of parents and the number of trees

Function	n	FDA	P/ T	S	Gen.
$Isotorus$	36	BOA	2	86	7.29
	36	BOA	4	81	7.23
	36	BOA	6	78	7.4
	36	Tree		72	7.96
	36	MT	2	87	8.22
	30	MT	4	89	7.83
	36	MT	6	93	7.38
$f_{3deceptive}$	30	BOA	2	78	7.47
	30	BOA	4	77	8.12
	30	BOA	6	77	8.49
	30	Tree		37	10.46
	30	MT	2	54	11.26
	30	MT	4	60	11.3
	30	MT	6	69	10.87

Table 3: Numerical results for MT-FDA and the BOA

for the BOA and the MT-FDA respectively. Column five represents the number of times the algorithm has converged in 100 runs. In the experiments for the $f_{3deceptive}$ function BOA clearly overperformed the MT-FDA. For this function two issues are worth to point out. The poor performance of the tree based FDA, and the observation that when the number of trees is increased the gap in the behavior between the MT-FDA and the BOA is reduced. Nevertheless, by further increasing the number of trees the MT-FDA does not achieve better results than BOA.

This situation is completely reversed for function Isotorus. For this function, as the number of trees is increased MT-FDA overperforms BOA, when the number of trees is 6 this difference is evident. The general conclusion from these experiments is that for certain kind of functions MT-FDA can overperform Bayesian-FDAs. One question remains open: which are the criteria that allow to decide whether it is more convenient to apply one FDA or another? We hypothesize MT-FDA can be more suitable for functions with general and scattered overlapping between the variables, that could be "covered" by a set of relatively dependent probabilistic models. This seems to be the case for the Isotorus function.

4.4 The role of the choice variable in the MT-FDA.

As it was briefly discussed in previous sections the use of the choice variable by the MT-FDA allows different and flexible ways of conducting the search. We conducted experiments to evaluate the convenience of using one of the variables of the problem as the choice

Function	n	N	z	T	S	Gen.
Nf^3_{dec}	31	2500	unknown	0.15	24	8.96
	46	3000	unknown	0.15	21	11.57
	61	3000	unknown	0.25	12	19.8
	31	1500	x_1	0.15	30	3.1
	46	1500	x_1	0.15	28	6.25
	61	2500	x_1	0.25	23	10.48

Table 4: Numerical results for the Nf^3_{dec} function.

variable. These experiments are very preliminary but help to illustrate how the parameters of the mixture model can be conveniently used in the FDA framework. Table 4 shows the results for function Nf^3_{dec}. This function has many global optimal that are triggered by variable x_1. When $x_1 = 1$ there is just one optimum (all variables are set to 1), when $x_1 = 0$ there $3^{\frac{n}{3}}$ optima (all possible combinations of 2 ones in every partition). For every setting 30 experiments were run using truncation selection and a maximum of 25 generations.

In the table it is shown a comparison between two MT-FDA with mixtures models composed of two trees. The mixture model of the first MT-FDA uses x_1 as its choice variable, for the second the choice variable is unknown as was the case in the previous experiments. When x_1 is treated as the choice variable the individuals that fulfill $(x_1 = 0)$ are approximated by one tree, and those that satisfy $(x_1 = 1)$ by the other. In this case there is not learning of the tree structures, only the mixture coefficients are calculated as $\lambda_0 = \frac{N_0}{N}, \lambda_1 = \frac{N_1}{N}$, where N_0 and N_1 are respectively the number of individuals that satisfy $(x_1 = 0)$ and $(x_1 = 1)$. Nevertheless such a mixture allows the MT-FDA to focus on the region of the space of solutions defined by the x_1 value. Eventually, as evolution advances, one of the coefficients becomes 0, and a tree based FDA is run from then on.

In Table 4 the improvements achieved by considering x_1 as the choice variable can be seen. This is a simple example of how the choice variable can be incorporated also as a tool for influencing the exploration and exploitation purposes of the search.

5 CONCLUSIONS AND FURTHER WORK

In this paper we have introduced a FDA based on mixture distributions. The MT-FDA is different to other simple connected based FDAs, and to Bayesian FDAs too. Contrary to Bayesian FDAs the MT model allows that the structure of dependencies among a set of variables changes. Compared to simple connected FDAs, in the MT-FDA, dependencies between the parents of a given node can be represented.

The algorithm exhibits a number of characteristics that places it far apart from Gallagher's algorithm (Gallagher, Frean, & Downs, 1999). It is appropriate for problems with integer representation that can have many variables. The performance of Gallagher's algorithm is more sensitive to the number of variables. The type of mixture model both algorithms employ is different, the adaptive mixture model of Priebe (Priebe, 1994) is used in Gallagher's algorithm.

Compared to the Estimation of Mixture of Distribution Algorithm (EMDA) introduced in (Peña, Lozano, & Larrañaga, 2001) the algorithm exhibits also several differences. MT-FDA is not focused on the optimization of multiobjective functions although it can be employed in multiobjective optimization too. Another difference is that our mixture model allows components with different structures. The version of the EM algorithm we use in this paper is different to the one presented in (Peña, Lozano, & Larrañaga, 2001).

In this paper we have shown that the MT-FDA can overperform the tree based FDA for the functions considered. The algorithm is also better than Bayesian FDAs for some functions like Isotorus. Another achievement of this paper is to have presented different ways to influence the search by manipulating parameters related to the mixture of trees.

The computational complexity of the MT-FDA is mainly given by the Estimation and Maximization steps of the (EM) algorithm. In each iteration of the mixture of trees learning algorithm the running time of the estimation step is $\Theta(mnN)$, and for the maximization step is $\Theta(mnr^2_{\max})$ where $r_{\max}$ is the maximum cardinality of the variables. Both steps are computationally expensive. We have not treated this question in the present work. Currently we are investigating how to reduce the computational burden associated to the EM algorithm in the context of the MT-FDA. Promising lines of research and further work are:

1. To design efficient learning schedules that help to diminish the number of evaluations.
2. To extend the use of mixtures to deal with more complex probabilistic models.
3. The use of MT-FDA as a method to implement parallel and distributed population based search.

Acknowledgments

We are grateful for constructive comments of anonymous reviewers. This work was funded by the Cuban Ministry of Science, Technology and Environment, under the project Low Cost Evolutionary Algorithms.

References

Baluja, S., and Davies, S. 1997. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proceedings of the 14th International Conference on Machine Learning*, 30–38. Morgan Kaufmann.

Chow, C. K., and Liu, C. N. 1968. Approximating discrete probability distributions with dependence trees. volume IT14, 462–467.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* B(39):1–38.

Etxeberria, R., and Larrañaga, P. 1999. Global optimization using Bayesian networks. In *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, 332–339.

Everitt, B., and Hand, D. 1981. *Mixture Models: Inference and Applications to Clustering*. London: Chapman and Hall.

Gallagher, M.; Frean, M.; and Downs, T. 1999. Real-valued evolutionary optimization using a flexible probability density estimator. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, 840–846. Orlando, FL: Morgan Kaufmann Publishers, San Francisco, CA.

Meila, M. 1999. *Learning Mixtures of Trees*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Mühlenbein, H., and Mahnig, T. 1999a. Evolutionary synthesis of Bayesian networks for optimization. *Evolutionary Computation* 7(1).

Mühlenbein, H., and Mahnig, T. 1999b. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation* 7(4):353–376.

Mühlenbein, H., and Mahnig, T. 2000. Evolutionary synthesis of Bayesian networks for optimization. *Advances in Evolutionary Synthesis of Neural Systems, MIT Press*. to be published.

Mühlenbein, H., and Paaß, G. 1996. From recombination of genes to the estimation of distributions I. Binary parameters. In Eiben, A.; Bäck, T.; Shoenauer, M.; and Schwefel, H., eds., *Parallel Problem Solving from Nature - PPSN IV*, 178–187. Berlin: Springer Verlag.

Mühlenbein, H.; Mahnig, T.; and Ochoa, A. 1999. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics* 5(2):213–247.

Ochoa, A.; Muehlenbein, H.; and Soto, M. R. 2000. A Factorized Distribution Algorithm using single connected Bayesian networks. In *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*. Paris, France: Springer Verlag. LNCS 1917.

Peña, J.; Lozano, J. A.; and Larrañaga, P. 2001. *Estimation Distribution Algorithms. A new tool for Evolutionary Optimization*. Boston/Dordrecht/London: Kluwer Academic Publishers. chapter Benefits of Data Clustering in Multimodal Function Optimization via EDAs, 99–124. To appear.

Pelikan, M., and Goldberg, D. E. 2000. Genetic algorithms, clustering, and the breaking of symmetry. IlliGAL Report No. 2000013, University of Illinois at Urbana-Champaign,Illinois Genetic Algorithms Laboratory, Urbana, IL.

Pelikan, M., and Mühlenbein, H. 1999. The Bivariate Marginal Distribution Algorithm. In Roy, R.; Furuhashi, T.; and Chawdhry, P., eds., *Advances in Soft Computing - Engineering Design and Manufacturing*, 521–535. London: Springer-Verlag.

Pelikan, M.; Goldberg, D. E.; and Cantú-Paz, E. 1999. BOA: The Bayesian Optimization Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, 525–532. Orlando, FL: Morgan Kaufmann Publishers, San Francisco, CA.

Pelikan, M.; Goldberg, D. E.; and Sastry, K. 2000. Bayesian Optimization Algorithm, decision graphs, and Occam's razor. IlliGAL Report No. 2000020, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Priebe, C. E. 1994. Adaptive mixtures. *Journal of the American Statistical Association* 89(427):796–806.

Soto, M. R.; Ochoa, A.; Acid, S.; and Campos, L. M. 1999. Bayesian evolutionary algorithms based on simplified models. In *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, 360–367.

Thierens, D., and Bosman, P. 2001. Multi-objective optimization with iterated density estimation evolutionary algorithms using mixtures models. In *Evolutionary Computation and Probabilistic Graphical Models. Proceedings of the Third Symposium on Adaptive Systems (ISAS-2001), Cuba*, 129–136.

Don't Evaluate, Inherit

Kumara Sastry, David E. Goldberg, and Martin Pelikan
Illinois Genetic Algorithms Laboratory, IlliGAL
Department of General Engineering
University of Illinois at Urbana-Champaign
104 S. Mathews Ave, Urbana IL 61801
{ksastry, deg, mpelikan}@uiuc.edu

Abstract

This paper studies fitness inheritance as an efficiency enhancement technique for genetic and evolutionary algorithms. Convergence and population-sizing models are derived and compared with experimental results. These models are optimized for greatest speed-up and the optimal inheritance proportion to obtain such a speed-up is derived. Results on OneMax problems show that when the inheritance effects are considered in the population-sizing model, the number of function evaluations are reduced by 20% with the use of fitness inheritance. Results indicate that for a fixed population size, the number of function evaluations can be reduced by 70% using a simple fitness inheritance technique.

1 Introduction

A key challenge in genetic and evolutionary computation (GEC) research is the design of *competent* genetic algorithms (GAs). By *competent* we mean GAs that can solve hard problems, quickly, reliably, and accurately, and much progress has been made along these lines (Goldberg, 1999). In essence competent GA design takes problems that were intractable with first generation GAs and renders them tractable, oftentimes requiring only a subquadratic number of fitness evaluations. But in large-scale problems, the task of computing even a subquadratic number of function evaluations can be daunting. This is especially the case if the fitness evaluation is a complex simulation, model, or computation. This places a premium on a variety of efficiency enhancement techniques. In this paper, one such efficiency enhancement technique called *fitness inheritance* is modeled and optimized for greatest speedup. In fitness inheritance, an offspring inherits a fitness value from its parents rather than through function evaluation.

The objective of this study therefore is to model fitness inheritance and to employ this model in predicting the convergence time and population size required for the successful design of a GA. We start by modeling fitness inheritance and deriving convergence time and population-sizing models. Subsequently, we derive an optimal proportion of inheritance and comment on the actual speed-up obtained. The speed-up that could be obtained under the practitioner's usual assumption of a fixed population sizing is also discussed.

2 Literature Review

Smith, Dike, and Stegmann (1995) proposed fitness inheritance in GAs. They proposed two ways of inheriting fitness, one by taking the average fitness and the other by taking a weighted average of the fitness of the two parents. They showed some theoretical justification for their approach. Their results indicated that GAs with fitness inheritance outperformed those without inheritance in both the OneMax and an aircraft routing problem. However, they did not investigate the effect of fitness inheritance on convergence time and population sizing. Also, the questions as to how many children should have inherited fitness, and how much speed-up one can get remained unanswered. Though the original study showed very encouraging results, unfortunately there have been very few follow up studies on fitness inheritance. Zheng, Julstrom, and Cheng (1997) used fitness inheritance for the design of vector quantization codebooks.

3 Modeling Fitness Inheritance

In the proposed approach, a proportion, p_i, of randomly selected individuals, receive inherited fitness and the rest are assigned the true (evaluated) fitness.

In the remainder of this paper, actual fitness refers to the fitness that a individual would have had if it was evaluated — that is, if its fitness was not inherited. In this section, we assume that the inherited fitness is taken to be the average of the building-block (BB) fitness. We assume this to develop a theory behind fitness inheritance and in the implementation the inherited fitness is taken to be the average fitness of the two parents. The building-block fitness is taken to be the average fitness of all the individuals in the population that possess the schemata under consideration. Also, in the remainder of the paper, unless otherwise mentioned, all the experimental results are obtained with crossover probability of 1.0.

The model derived is applicable to uniformly scaled problems of fixed string length and known BB size. Specifically OneMax (counting of bits) is employed but the model can be extended to other problems in a straightforward manner. We further assume that the actual fitness distribution, F, is Gaussian with mean $\mu_{f,t}$ and variance $\sigma^2_{f,t}$.

$$F = N\left(\mu_{f,t}, \sigma^2_{f,t}\right),$$

and that the distribution of fitness with inheritance, F′ is Gaussian with mean $\mu_{f',t}$ and variance $\sigma^2_{f',t}$.

$$F' = N\left(\mu_{f',t}, \sigma^2_{f',t}\right).$$

The above assumptions are justified since crossover has a normalizing effect. We can write

$$\begin{aligned} \mu_{f',t} &= \mu_{f,t}(1-p_i) + \mu_{i,t}p_i, & (1)\\ \sigma^2_{f',t} &= (1-p_i)\sigma^2_{f,t} + p_i\sigma^2_{i,t}, & (2) \end{aligned}$$

where $\mu_{i,t}$, and $\sigma^2_{i,t}$ are the mean and variance of fitness respectively, of individuals whose fitness is inherited. Since the inherited fitness, f_i, is equal to the average of BB fitness we can write

$$f_i = \frac{1}{\ell}\sum_{j=1}^{\ell} \hat{f}\left(BB_j\right), \qquad (3)$$

where, ℓ is the string length, and $\hat{f}\left(BB_j\right)$ is the estimated BB fitness which can be written

$$\hat{f}\left(BB_j\right) = f\left(BB_j\right) + (\ell-1)p, \qquad (4)$$

where, $f\left(BB_j\right)$ is the actual BB fitness, p is the proportion of correct BBs, and the term $(\ell-1)p$ incorporates the noise arising from other BBs. Using the above relation, f_i for uniformly scaled problems can be written as

$$f_i = \frac{f}{\ell} + (\ell-1)p. \qquad (5)$$

The mean inherited fitness, $\mu_{i,t}$ is given by

$$\begin{aligned} \mu_{i,t} &= \frac{1}{n}\sum_{j=1}^{n} f_{i,j},\\ &= \frac{1}{n}\sum_{j=1}^{n}\left[\frac{f_j}{\ell} + (\ell-1)p\right],\\ &= \frac{1}{\ell}\left[\frac{1}{n}\sum_{j=1}^{n} f_j\right] + (\ell-1)p,\\ &= \ell p = \mu_{f,t}, & (6) \end{aligned}$$

where n is the population size. Using the above relation in equation 1, we get

$$\mu_{f',t} = \mu_{f,t}. \qquad (7)$$

The inherited fitness variance, $\sigma^2_{i,t}$ can be derived as follows,

$$\begin{aligned} \sigma^2_{i,t} &= \frac{1}{n}\sum_{j=1}^{n} f^2_{i,j} - (\ell p)^2,\\ &= \frac{1}{n}\sum_{j=1}^{n}\left[\frac{f_j}{\ell} + (\ell-1)p\right]^2 - (\ell p)^2,\\ &= \frac{p(1-p)}{\ell}. & (8) \end{aligned}$$

Using the above relation in equation 2 we get,

$$\begin{aligned} \sigma^2_{f',t} &= (1-p_i)\sigma^2_{f,t} + p_i\frac{p(1-p)}{\ell}\\ &\approx (1-p_i)\sigma^2_{f,t} & (9) \end{aligned}$$

Using the notion of *selection intensity*, I (Bulmer, 1980), we can write the expected average fitness with inheritance after selection as

$$\begin{aligned} \mu_{f',t+1} &= \mu_{f',t} + I\sigma_{f',t},\\ &= \mu_{f',t} + I\sqrt{1-p_i}\sigma_{f,t}. & (10) \end{aligned}$$

Since both the actual fitness and inherited fitness distributions are normally distributed, a bivariate normal distribution can be used to obtain the expected actual fitness value of F at generation t+1, given $\mu_{f',t+1}$,

$$E(F/\mu_{f',t+1}) = \mu_{f,t+1} = \mu_{f,t} + \frac{\sigma_{F,F'}}{\sigma^2_{f',t}}(\mu_{f',t+1} - \mu_{f',t}).$$

It can be easily seen that the covariance, $\sigma_{F,F'}$ is $(1-p_i)\sigma^2_f$. Using this relation and equation 10,

$$\begin{aligned} \mu_{f,t+1} &= \mu_{f,t} + (\mu_{f',t} + I\sqrt{1-p_i}\sigma_{f,t} - \mu_{f',t}),\\ &= \mu_{f,t} + I\sqrt{1-p_i}\sigma_{f,t}. & (11) \end{aligned}$$

We now proceed to derive the convergence and population-sizing models.

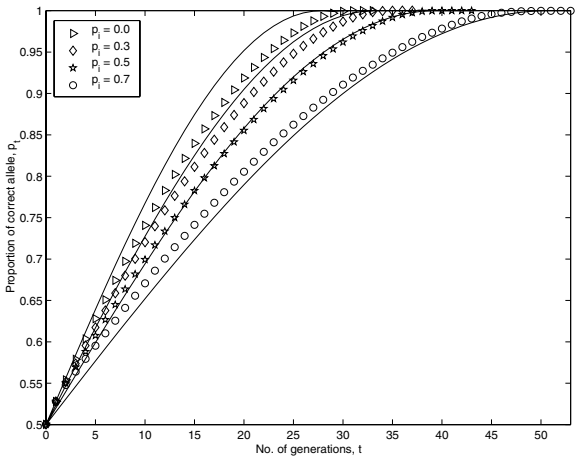


Figure 1: Verification of proportion of correct BBs predicted by equation 13 with empirical results plotted as a function of generation number for different values of inheritance proportion. The experimental results are averaged over 50 runs. The discrepancy between the theoretical and experimental results are due to hitchhiking and can be eliminated by ensuring a better mixing of BBs through repeated crossover or population-wise crossover.

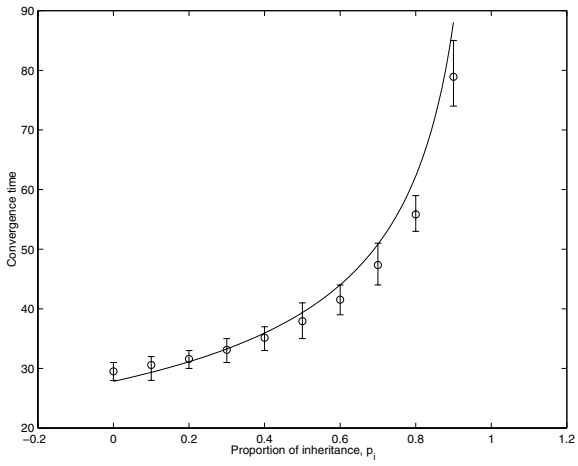


Figure 2: Convergence time for a 100-bit OneMax problem for different proportion of inheritance predicted by equation 14 compared to experimental results. The empirical results are averaged over 50 runs.

3.1 Time to Convergence

In this section we derive convergence model for the OneMax problem with fitness inheritance. For OneMax domain, we can write

$$\mu_{f,t} = \ell p_t, \quad \sigma^2_{f,t} = \ell p_t(1-p_t),$$

where, ℓ is the string length, and p_t is the proportion of correct alleles in the population at generation t. Since the initial population is generated with uniform distribution, $p_0 = 0.5$. Using the above relation in equation 11,

$$\begin{aligned} p_{t+1} &= p_t + I\sqrt{\frac{(1-p_i)}{\ell}}\sqrt{p_t(1-p_t)}, \\ p_{t+1} - p_t &= I\sqrt{\frac{(1-p_i)}{\ell}}\sqrt{p_t(1-p_t)}. \end{aligned}$$

Approximating the above equation as a differential equation yields

$$\frac{dp}{dt} = \frac{I\sqrt{1-p_i}}{\sqrt{\ell}}\sqrt{p(1-p)}. \tag{12}$$

Integrating the above equation and using the initial condition $p|_{t=0} = 0.5$ we get,

$$p_t = \sin^2\left(\frac{\pi}{4} + \frac{I\sqrt{(1-p_i)}t}{2\sqrt{\ell}}\right). \tag{13}$$

The above equation is compared to the experimental results in figure 1 at different inheritance proportions p_i. The proposed convergence model slightly overestimates the proportion of correct BBs for GAs with tournament selection and uniform crossover. This discrepancy between the theoretical and empirical results can be eliminated by employing recombination procedures than ensure that the BBs are well mixed (Thierens, 1995).

We can derive an equation for convergence time, t_{conv}, by equating $p_t = 1$, and inverting equation 13,

$$t_{\text{conv}} = \frac{\pi}{2I}\sqrt{\frac{\ell}{(1-p_i)}}. \tag{14}$$

If p_i is taken as 0 then the above relation reduces to $\pi\sqrt{\ell}/(2I)$ which agrees with existing convergence time models (Muhlenbein & Schlierkamp-Voosen, 1993; Miller & Goldberg, 1996a; Miller & Goldberg, 1996b). The convergence time observed experimentally is compared to the above prediction for a 100-bit OneMax problem in figure 2. Again the discrepancy between the empirical and analytical results occurs due to hitch-hiking and can be reduced by ensuring a good mixing of building blocks.

3.2 Population Sizing

It is well known that population size is a major determinant of the quality of the solution obtained. Therefore it is essential to appropriately size the population to incorporate the effects of fitness inheritance. Goldberg, Deb, and Clark (1992) proposed population-

sizing models for different selection schemes. Their model is based on deciding correctly between the best and the second best BBs in the same partition. They incorporated noise arising from other partitions into their model. However, they assumed that if wrong BBs were chosen in the first generation, the GAs would be unable to recover from the error. Harik, Cantu-Paz, Goldberg, and Miller (1997) refined the above model by incorporating cumulative effects of decision making over time rather than in first generation only. They modeled the decision making between the best and second best BBs in a partition as a gambler's ruin problem. This model is based on the assumption that the selection process used is tournament selection without replacement. Miller (Miller, 1997) extended this model to predict population sizing in the presence of external noise. The population-sizing model derived by Miller is reproduced below.

$$n = -\frac{2^{k-1}\log(\psi)\sqrt{\pi}}{d_{\min}}\sqrt{\sigma^2_{f'}},$$

where n is the population size, k is the BB length, ψ is the failure rate, $d_{\min}$ is the distance between the best BB and the second best BB (Goldberg, Deb, & Clark, 1992), and $\sigma^2_{f'}$ is the variance of the noisy fitness function. Not only $\sigma^2_{f'}$, but also $d_{\min}$ depends on p_i. For OneMax problems $d_{\min}$ was empirically determined to be

$$d_{\min} = (1-p_i^3)\sqrt{1-p_i}. \tag{15}$$

The population-sizing equation can now be written as

$$n = -\frac{2^{k-1}\log(\psi)\sqrt{\pi}}{(1-p_i^3)}\sqrt{\sigma^2_f}. \tag{16}$$

The above population sizing is compared to the results obtained for a 100-bit OneMax problem in figure 3. From the plot we can easily see that our population-sizing model fits the experimental result accurately. Using the convergence time and population-sizing model derived in this section, we evaluate the inheritance proportion that requires least number of function evaluations (or equivalently, yields greatest speed-up) in the next section.

4 Optimal Inheritance Proportion

We can intuit that given a problem there should be a value (or a range) of inheritance proportions that are more efficient than the others. Too low a p_i or too high a p_i would not reduce the number of function evaluations. Our aim is to determine the inheritance proportion such that the total number of function evaluation required is minimized. Here we assume that the

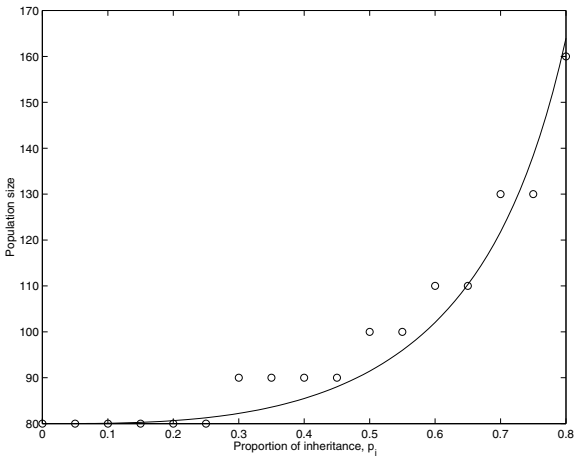


Figure 3: Verification of the population-sizing model (equation 16) for various inheritance proportions with empirical results. Experimental results display the population size required for optimal convergence with failure rate of 0.001 and are averaged over 50 runs.

cost of inheritance is insignificant. This is justified by the fact that inherited fitness is just an average of the fitness values of the two parents, a computationally trivial task when compared to the usual function evaluation. We reiterate that fitness inheritance is needed in cases where function evaluation takes a long time (eg., a large real-world problem), or when only some individuals can be evaluated (for example, interactive GAs). Total number of function evaluations required is given by

$$\begin{aligned} N_{fe} &= n\left[(t_{\text{conv}}-1)(1-p_i)+1\right], \\ &= n\left[t_{\text{conv}}(1-p_i)+p_i\right]. \end{aligned} \tag{17}$$

From previous sections, for a given problem,

$$\begin{aligned} t_{\text{conv}} &= \frac{c_2}{\sqrt{1-p_i}}, \\ n &= \frac{c_3}{1-p_i^3}, \end{aligned}$$

where c_2 and c_3 are $\pi\sqrt{\ell}/(2I)$ and $-2^{k-1}\log(\psi)\sqrt{\pi\sigma^2_f}$ respectively. Using the above equations, the total number of function evaluations is given by,

$$N_{fe} = \frac{c_3}{1-p_i^3}\left[c_2\sqrt{1-p_i}+p_i\right]. \tag{18}$$

The optimal proportion of inheritance is then given by solving,

$$\begin{aligned} \frac{\partial N_{fe}}{\partial p_i} &= 0, \\ 3p_i^2\left[c_2(1-p_i)+p_i\sqrt{1-p_i}\right] + & \\ (1-p_i^3)\left[-0.5c_2+\sqrt{1-p_i}\right] &= 0. \end{aligned} \tag{19}$$

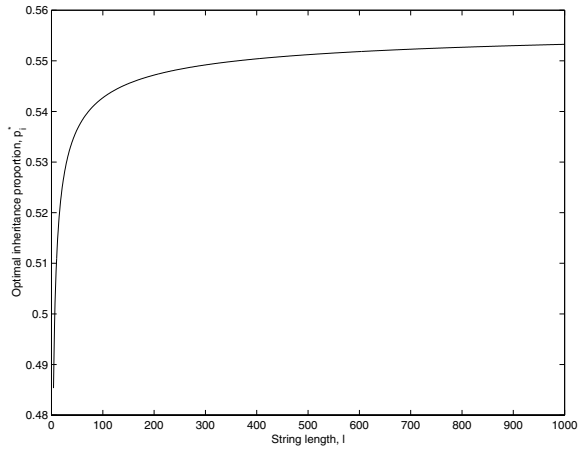


Figure 4: Optimal inheritance proportion, p_i^* as a function of string length ℓ obtained by numerical solution of equation 19. p_i^* is independent of ℓ for moderate to large values of ℓ.

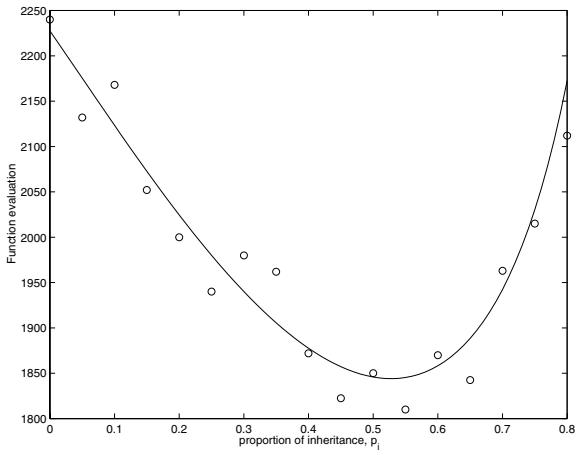


Figure 5: Total number of function evaluations predicted by equation 17 is compared to empirical results as a function of inheritance proportion. The experimental results display the total number of function evaluations required for optimal convergence of a 100-bit OneMax problem with a failure rate of 0.0001. The empirical results are averaged over 50 runs.

The above equation can be solved for two asymptotic cases: (1) the string length, $\ell = 0$, then $c_2 = 0$, and the optimal evaluates to $p_i^* = 0$, and (2) the string length is very long, then $c_2 \to \infty$. For this case equation 19 reduces to

$$3p_i^2\left(1-p_i\right) - \frac{1}{2}\left(1-p_i^3\right) = 0, \quad (20)$$

$$p_i^2 - \frac{1}{5}p_i - \frac{1}{5} = 0. \quad (21)$$

The above quadratic equation can be easily solved, and the optimal proportion for this case comes out to be $p_i^* = 0.558$. For other values of string length, equation 19 cannot be solved analytically, and hence it has been solved numerically for different problem sizes. The optimal proportion of inheritance obtained by solving the above equation numerically is plotted as a function of string length is shown in figure 4. We can see that for moderate to large sized problems the optimal proportion of inheritance, p_i^* lies between 0.54–0.558, that is,

$$0.54 \leq p_i^* < 0.558 \quad (22)$$

The above result (equation 22) suggests that p_i is independent of problem size for problems of moderate to large size. The predicted number of function evaluations is compared with experimental results for a 100-bit OneMax in figure 5, for a 40-bit trap function with BB size 4 with a crossover probability of 0.9 in figure 6(a), and for a 40-bit trap function with BB size 4 with tournament size of 8 and crossover probability of 1 in figure 6(b). Even though our model was derived for OneMax problems, it holds even for other problems with different parameter settings as shown by the plots. This exemplifies the robustness and usefulness of the proposed model. Another point to be noted is that the optimal inheritance proportion is between 0.54–0.558 in all cases.

Another interesting fact to note is that the number of function evaluations with inheritance is only around 20% less than that without inheritance. In other words the speed-up defined as the ratio of number of function evaluations with $p_i = 0$ to the number of function evaluations at optimal p_i is around 1.2. This implies that we get a moderate advantage by using fitness inheritance. The existence of an optimal p_i and the moderate value of speed-up are in contrast with the earlier studies on inheritance. A detailed discussion of this discrepancy is presented in the next section.

5 Apparent Speed-up

In the previous section we presented the speed-up that can be obtained if the population size is chosen appropriately. This is the speed-up that theoreticians can obtain when they adjust conditions appropriately to hold the solution quality constant. A GA practitioner, unlike a GA theoretician, views GAs as means to reach an end. He usually fixes the population size and then opts for fitness inheritance. From a GA practitioner's point of view, the speed-up obtained through fitness inheritance can be much higher. We call this speed-up, that is obtained through a fixed population size as

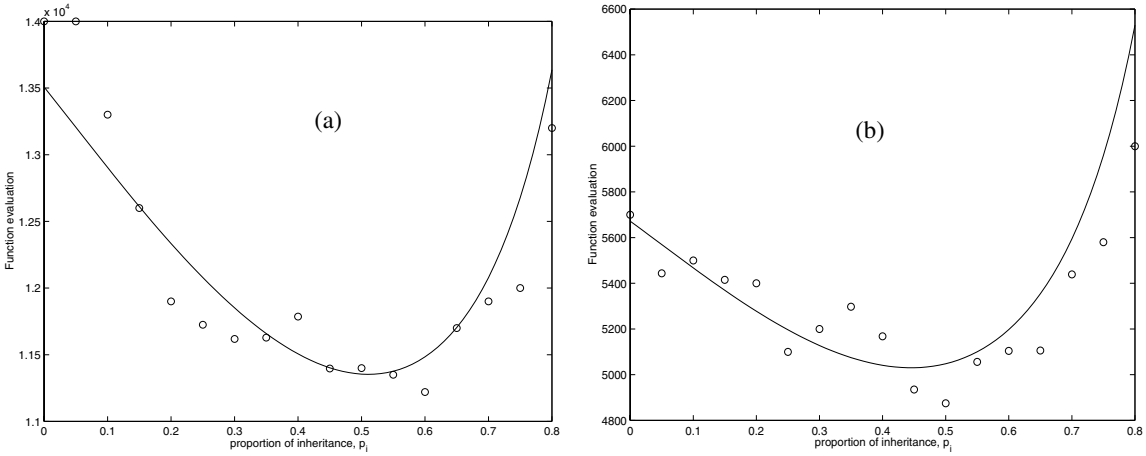


Figure 6: Total number of function evaluations predicted by equation 17 is compared to empirical results as a function of inheritance proportion. The experimental results display the total number of function evaluations required for optimal convergence of a 40-bit trap function with a BB size of 4, and a failure rate of 0.0001. The empirical results are averaged over 50 runs. Crossover probability is (a) 1.0, and (b) 0.8. The results indicate that the optimal proportion of inheritance given by equation 22 is in fact approximately valid for other uniformly scaled problems with BB size greater than one and different GA parameter values.

apparent speed-up.

Function evaluations taken for different population sizes plotted as a function of p_i for a 100-bit OneMax problem is shown in figure 7. The plot indicates only those points for which the population converged to the optimal solution in all 50 runs. The apparent optimal inheritance proportion, p_i^{app}, is given by the inverse of the population-sizing model, equation 16.

$$p_i^{\text{app}} = \sqrt[3]{1 - \frac{\kappa}{n}}, \tag{23}$$

where κ is a constant dependent on the problem type, and the solution quality desired, and is related by $\kappa = -2^{k-1}\log(\psi)\sqrt{\pi}$. There are two asymptotic cases for the above result. One, when the population size is less than κ, then fitness inheritance does not yield any speed-up and in fact can result in premature convergence. The other case is when the population size is very large when compared to κ. In this case a very high inheritance proportion can be used and high speedup values can be obtained.

The apparent optimal inheritance proportion predicted by equation 23 is compared to experimental results for a 100-bit OneMax in figure 8. The value κ for this problem is 81.63. The experimental results indicate the inheritance proportion that required lowest number of average function evaluations to converge to optimal solution in all 50 runs. From figure 7, it can be seen that if we choose an arbitrarily high population size, say 300, then fitness inheritance can yield a

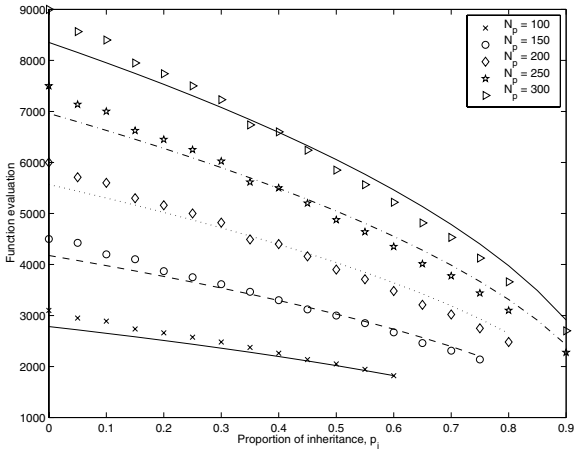


Figure 7: Total number of function evaluations for various proportions of inheritance at different population sizes. The experimental results are averaged over 50 runs and are compared to the results predicted using equation 17. Experimental results include only those points for which all 50 runs converged to the optimal solutions.

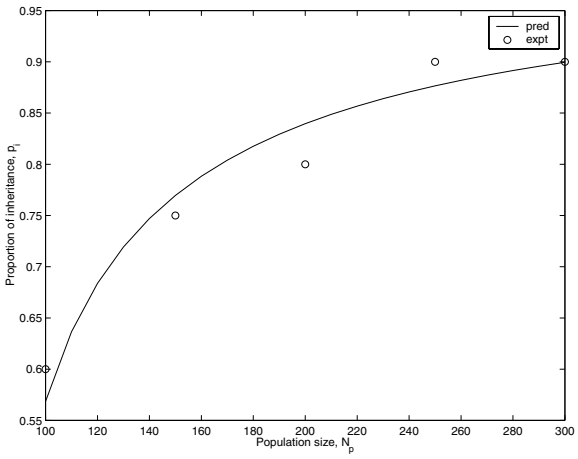


Figure 8: Apparent optimal inheritance proportion, p_i^{app}, predicted by equation 23 compared to empirical results. The empirical results display the inheritance proportion that requires minimum number of function evaluations to converge to the optimal solution for a 100-bit OneMax problem. The experimental results are averaged over 50 runs. The value of κ is 81.63.

speed-up of around 3.3. If we have a still higher population size, then the speed-up will be higher. This result agrees with that obtained by Smith, Dike, and Stegmann (1995), in which they had considered a 64-bit OneMax problem and had taken a population size of 500. A 100-bit OneMax without inheritance requires a population size of about 80 which implies that a 64-bit problem would need a still lower population size. The reason why Smith, Dike, and Stegmann (1995) did not get an optimal proportion of inheritance was due to the fact that they took a very high population size and did not compare the minimum population size required for different proportions of inheritance. In other words, they did not consider the effect of fitness inheritance on population sizing. Of course, GA practitioners are likely to do so, and our theories are able to explain the large apparent speed-up they shall achieve.

6 Future Work

In the present study we have analyzed fitness inheritance for OneMax problems and the proposed model can be extended to other problems (eg., non-uniformly scaled problems). Further investigation is required for determining analytically the signal to noise-ratio used in the population-sizing model. The inheritance procedure used in the present study is a simple one, and a study on more complex inheritance techniques still remains to be done. The present analysis is developed for the OneMax problem, which is a GA easy problem. Therefore, the speed-up obtained in the current study is an upper bound and we recognize that a lower speed-up could be obtained for more complex or GA hard problems. The greatest speed-ups obtained for such cases have to be investigated.

7 Conclusions

In this paper, we have developed a theoretical basis for fitness inheritance and have derived models for convergence time and population sizing. These results have been integrated into a model that predicts solution quality and cost, and this model has been analyzed and optimized for greatest speed-up. Under careful conditions of adjusting GA parameters for constant solution quality, the optimum inheritance yields savings of 20% in the number of function evaluations. Though by itself this speed-up value seems to be modest, it can be coupled with parallelism, time continuation, and other evaluation relaxation schemes. In such a scenario the effective speed-up obtained will be a product of all individual speed-ups and even a speed up of 1.2 can be important. We have been careful to include the effects of fitness inheritance on quality-duration theory in predicting the above results. However, GA practitioners usually fix the population size and then try inheritance. Under these conditions the apparent speed-up can be much greater, a result that agrees with the earlier empirical study of Smith, Dike, and Stegmann (1995).

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0163. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.

Thanks are also due to Fernando Lobo, Ravi Srivas-

tava, and Abhishek Sinha for their useful comments.

References

Bulmer, M. (1980). *The mathematical theory of quantitative genetics.* Oxford: Clarendon Press.

Goldberg, D. (1999). The race, the hurdle, and the sweet spot: Lessons from genetic algorithms for the automation of design innovation and creativity. *Evolutionary Design by Computers*, 105–118.

Goldberg, D., Deb, K., & Clark, J. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Harik, G., Cantu-Paz, E., Goldberg, D., & Miller, B. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In Back, T., et al. (Eds.), *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 7–12). Piscataway, NJ, USA: IEEE.

Miller, B., & Goldberg, D. (1996a). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131.

Miller, B. L. (1997, May). *Noise, sampling, and efficient genetic algorithms.* Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

Miller, B. L., & Goldberg, D. E. (1996b). Genetic algorithms, tournament selection, and the varying effects of noise. *Complex Systems*, *9*(3), 193–212.

Muhlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. continous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Smith, R., Dike, B., & Stegmann, S. (1995). Fitness inheritance in genetic algorithms. In *Proceedings of the ACM Symposium on Applied Computing* (pp. 345–350). New York, NY, USA: ACM.

Thierens, D. (1995). *Analysis and design of genetic algorithms.* Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.

Zheng, X., Julstrom, B., & Cheng, W. (1997). Design of vector quantization codebooks using a genetic algorithm. In *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC* (pp. 525–529). Picataway, NJ, USA: IEEE.

On the Importance of the Second Largest Eigenvalue on the Convergence Rate of Genetic Algorithms

Florian Schmitt and Franz Rothlauf*
Department of Information Systems
University of Bayreuth
Universitaetsstr. 30
D-95440 Bayreuth/Germany
florian.schmitt@stud.uni-bayreuth.de, rothlauf@uni-bayreuth.de

Abstract

Genetic algorithms are sometimes disparagingly denoted as just a fancier form of a plain, stupid heuristic. One of the main reasons for this kind of critique is that users believed a GA could not guarantee global convergence in a certain amount of time.

Because the proof of global convergence of GAs using elitism has been performed elsewhere (13), in this work we want to extend previous work by J. Suzuki (15) and focus on the identification of the determinants that influence the convergence rate of genetic algorithms. The convergence rate of genetic algorithms is addressed using Markov chain analysis. Therefore, we could describe an elitist GA using mutation, recombination and selection as a discrete stochastic process. Evaluating the eigenvalues of the transition matrix of the Markov chain we can prove that the convergence rate of a GA is determined by the second largest eigenvalue of the transition matrix. The proof is first performed for diagonalizable transition matrices and then transferred to matrices in Jordan normal form.

The presented proof allows a more detailed and deeper understanding of the principles of evolutionary search. As an extension to this work we want to encourage researchers to work on proper estimations of the second largest eigenvalue of the transition matrix. With a good approximation, the convergence behavior of GAs could be described more exactly and GAs would be one step ahead on the road to a fast, reliable and widely accepted optimization method.

*Also with Illinois Laboratory of Genetic Algorithms, University of Illinois at Urbana-Champaign, USA.

1 Introduction

Sometimes researchers speak disparagingly about genetic algorithms and label them to be just fancier form of a plain, simple heuristic. One main reason for this is that genetic algorithms (GA) stick to the prejudice that they are not able to guarantee convergence to the global optimum. The users do not know if the GA converges for a specific problem to the global optimum, and how much time the GA needs to converge. A closer look at genetic algorithms, however, reveals that there exists not only proof of the global convergence for genetic algorithms using elitism (13), but also some work about the convergence rate of GAs (15). The more complicated analysis of the convergence rate is important because a genetic algorithm which can be proven to converge, but needs infinite time for it, is not helpful for a effective use of genetic algorithms and would confirm the prejudices against GAs.

In this work we want to perform a more detailed analysis of the convergence rate using Markov chains. The Markov chain model is used for modeling a simple GA with the genetic operators selection, mutation and crossover. We investigate the determinants which the convergence rate depends on. With using some results from G. Rudolph (13) and J. Suzuki (15) we can prove that the convergence rate depends mainly on the value of the second largest eigenvalue of the transition matrix of the Markov chain. Furthermore, we transfer the results we get for diagonalizable transition matrices to the more general class of matrices in Jordan normal form. We illustrate that the proof for the convergence rate holds true for matrices in Jordan normal form, too.

The paper is structured as follows. In the following section we review some of the previous work about convergence behavior of genetic algorithms. This is followed in section 3 by presenting the requisites we want to use for our mathematical proof. We illustrate

the assumptions for the genetic algorithm, present some fundamentals about the used Markov chain models, and review some properties of stochastic matrices. The section ends with a lemma about stochastic matrices from M. Iosifescu (10) and the global convergence statement from G. Rudolph (13). In section 4 we present the proof that the convergence rate is mainly determined by the second largest eigenvalue of the Markov chain describing a GA. First, the proof is performed for diagonalizable matrices and then transferred to matrices in Jordan normal form. The paper ends with concluding remarks.

2 Previous work

In this section we give a short review about the two main approaches that are used for investigating the convergence behavior of genetic algorithms.

In the field of genetic algorithms and convergence behavior we could distinguish two large areas of research. The first line of research results from the theoretical investigations by J. Holland (9) and D. E. Goldberg (6) and are based on the schema theorem and the existence of building blocks for selectorecombinative genetic algorithms. In depth work in this line of research was done by H. Mühlenbein (2), D. Thierens (17), or D.E. Goldberg (7). A comprehensive overview of convergence time complexity can be found in D. Thierens thesis (16). All these models use the notion of building blocks, and are able to describe how building blocks grow and how long it takes to overtake a population accurately.

The second line of research treats GAs as a stochastic process with special properties. Starting with D.E. Goldberg (8) using Markov chains for modeling a GA, further work was done by T.E. Davis (3), A.E. Nix (11), and M.D. Vose (18) showing that a GA using selection, mutation and crossover can be fully described by the transition matrix of a Markov chain. Markov chains were also used by A.E. Eiben et al., G. Rudolph, and A. Agapie for the proof of the global convergence of evolutionary algorithms (4; 13; 1). The proof was an important step towards a better theoretical understanding of GAs . Based on the global convergence proof, J. Suzuki (15) identified the influence of the eigenvalues of the transition matrix of the Markov chain on the convergence rate of a GA. Although he gave upper and lower bounds for the convergence rate, he was not able to specify the important eigenvalue for the convergence rate exactly.

3 Preliminaries

This section provides the background that is necessary for understanding the investigations concerning the convergence analysis. We start by defining some basic properties of the used GA. This is followed by a description of the basic concepts of Markov chains and a review of the properties of stochastic matrices. The section ends with the proof of GA convergence as provided in (13).

3.1 Properties of the Genetic Algorithm

This paper deals with a kind of simple Genetic Algorithm (GA) in which the genetic operators are restricted to crossover, mutation and selection. Furthermore, the GA uses a binary representation of fixed length. The population size of the GA is determined a priori, and the probabilities for the three operators are not equal to zero.

For our investigation we use elitism in a way that the best parent survives if it is better than the best offspring. The individual with the highest fitness among all possible individuals is denoted as *super individual*. It represents the global optimum of the problem.

Further assumptions concerning the fundamental structure of the GA are not necessary in this context. Different types of crossover and mutation operators should have no influence on the convergence behavior of GAs and the proof shown in section 4 should still hold.

3.2 Markov chain analysis

The principal behavior of a GA can be described by using the Markov chain model. Using this concept we are able to develop a convergence model for the GA.

A Markov chain is a discrete stochastic process. The behavior of the stochastic process in future states depends only on the present states, but not on the past ones. Therefore, the probabilistic motion of a Markov chain could be described by using a transition matrix P.

For homogeneous Markov chains the t-th step transition matrix P^t can be determined iteratively. The Chapman-Kolmogorov equations (compare M. Iosifescu, p.65 (10)) yield

$$P^t = \prod_t P.$$

Let p_i^t denote the probability that the Markov chain is in state i at step t. The p_i^t can be gathered in a row

vector $p^t = (p_1^t, p_2^t, \ldots, p_n^t)$. The initial distribution p^0 is similarly defined. Then

$$p^t = p^0 \cdot P^t$$

for $t \geq 0$. Therefore, a homogeneous Markov chain is completely determined by the tuple (p^0, P).

The distribution p on the states of the Markov chain is called a stationary distribution, if $pP = p$, and is called a limit distribution, if the limit $p = p^0 \lim_{t\to\infty} P^t$ exists.

Every transition matrix of a Markov chain is stochastic. A non-negative matrix is said to be *stochastic* if all its row sums are equal to one. Further matrix classifications occurring in the following are given by G. Rudolph, p.55 (13). Stochastic matrices possess special properties:

- The eigenvalues of a stochastic matrix have modulus less or equal to 1.
- An irreducible stochastic matrix possess a simple unit eigenvalue.
- The right-hand eigenvector corresponding to a unit eigenvalue of a stochastic matrix is given by $e = (1, \ldots, 1)^T$.
- The vector p is a stationary probability vector of a stochastic matrix, if a left-hand eigenvalue corresponds to a unit eigenvalue.

The source of these statements can be found in W. Stewart, p. 28-30 (14).

3.3 Proof of global convergence

A qualitative Markov chain model of GAs is sufficient for the global convergence proof. The following lemma is necessary for the convergence proof presented by G. Rudolph (13). We use this lemma later for the analysis of the convergence rate.

Lemma 1
Let P be a reducible stochastic matrix, where $C \in \mathbb{R}^{m\times m}$ is a primitive stochastic matrix and $R, T \neq 0$. Then

$$\begin{aligned} P^\infty &= \lim_{t\to\infty} P^t \\ &= \lim_{t\to\infty} \begin{pmatrix} C^t & 0 \\ \sum_{i=0}^{t-1} T^i R C^{t-i} & T^t \end{pmatrix} \\ &= \begin{pmatrix} C^\infty & 0 \\ R^\infty & 0 \end{pmatrix} \end{aligned}$$

is a stable stochastic matrix with $P^\infty = e\,p^\infty$, where $p^\infty = p^0 P^\infty$ is unique regardless of the initial distribution, and the limit distribution p^∞ satisfies

$$p_i^\infty > 0 \quad \text{for } 1 \leq i \leq m \text{ and } p_i^\infty = 0$$

for $m < i \leq n$.

Matrix C is associated with the absorbing states of the Markov chain. For the proof see M. Iosifescu, p.126 (10). Using this lemma we can finally present the global convergence statement:

A genetic algorithm with an arbitrary initial distribution converges to the global optimum if the following assumptions are fulfilled:

- Selection chooses the best individual from parents and offspring (elitism).
- Every state is reachable from any other state.

For a detailed proof the reader is referred to G. Rudolph, chapter 5 (13).

4 Analysis of the convergence rate

For an analysis of the convergence rate, the proof of convergence, illustrated in the previous section, is a necessary condition. If we can not prove that the GA converges an investigation into convergence rate is useless. Using the convergence proof we prove in this section that the convergence rate is mainly determined by the second largest eigenvalue of the Markov chain describing a GA.

The theoretical identification of the second largest eigenvalue as the fundamental parameter influencing the convergence rate is first obtained for diagonalizable matrices. This result is then transferred to matrices in Jordan normal form, which represents a more general class of matrices.

4.1 Diagonalizable matrices

We want to start by identifying a measurement for the convergence rate of a GA. This should allow us to determine the progress of the GA's convergence.

J. Suzuki (15) measures the convergence rate as the degree in which the individual with the highest fitness in a population coincides with the super individual. Therefore, he investigates how closely the probability $\sum_{k\in X^*} p_k^n$ converges to $\sum_{k\in X^*} p_k^\infty$, where X^* denotes

the amount of populations containing the super individual. We want to use the same convergence measurement for the following analysis of the convergence rate.

Due to the necessary assumption that the GA is globally convergent, the probability $\sum_{k \in X^*} p_k^\infty$ converges to one (Lemma 1). Based on Suzuki's measurement of convergence and Lemma 1 we analyze how fast the probability $\sum_{k \in X^*} p_k^n$ converges to one for a finite number of generations.

Furthermore, we use for the analysis of the convergence rate of transition matrices the classical Perron Formula (compare V. Romanovsky, chapter 1 (12)). This formula allows us to compute the powers of a square matrix P. In the case of diagonalizable matrices, the Perron Formula can be reduced to

$$P = \sum_{i=1}^{n} \lambda_i v_i u_i^T, \tag{1}$$

where λ_i is an eigenvalue of matrix P, v_i and u_i are the corresponding right and left eigenvectors. Equation 1 is called the spectral representation of a matrix. As a relevant consequence it follows

$$P_{k,\nu}^n = \sum_{i=1}^{N} w_{k,\nu}^{(i)} \lambda_i^n. \tag{2}$$

Using the previous statements we could finally formulate the theorem that the convergence rate depends on the second largest eigenvalue of the diagonalizable transition matrix:

Theorem 1
Let $C > 0$ be constant.
A constant C exists which satisfies

$$\sum_{k \in X^*} p_k^n \geq 1 - C \cdot |\lambda_2|^n, \tag{3}$$

where $|\lambda_2|$ is the second largest eigenvalue of a diagonalizable transition matrix P describing the Markov chain of a global convergent GA.

Proof of Theorem 1
If $k \in X^*$, then can be followed:

$$\begin{aligned} P_{k,\nu}^n &= 0 \qquad \nu \neq k, \\ P_{k,\nu}^n &= 1 \qquad \nu = k. \end{aligned}$$

Using equation (2) the following equation holds for $k \notin X^*$

$$P_{k,\nu}^n = \sum_{i=1}^{N} w_{k,\nu}^{(i)} \lambda_i^n.$$

P is a stochastic matrix. Therefore, the largest eigenvalue of P is equal to unity. The simple unit eigenvalue is identified as the absorbing state of the corresponding Markov chain. Furthermore,

$$w_{k,\nu}^{(1)} = (v_1 u_1^T)_{k,\nu} = e \cdot p^{\infty^T} = \begin{pmatrix} p_1^\infty & \cdots & p_N^\infty \\ \vdots & & \vdots \\ p_1^\infty & \cdots & p_N^\infty \end{pmatrix}$$

holds because of the properties implied by a stochastic matrix (compare subsection 3.2). As P describes a Markov chain, we get

$$\begin{aligned} p_\nu^n &= \sum_{k=1}^{N} p_k^0 P_{k,\nu}^n \\ &= \sum_{k=1}^{N} p_k^0 \sum_{i=1}^{N} w_{k,\nu}^{(i)} \lambda_i^n. \end{aligned}$$

Therefore, $\sum_{\nu \notin X^*} p_\nu^n =$

$$\begin{aligned} &= \sum_{\nu \notin X^*} \sum_{k=1}^{N} p_k^0 \sum_{i=1}^{N} w_{k,\nu}^{(i)} \lambda_i^n \\ &= \sum_{\nu \notin X^*} \sum_{k=1}^{N} p_k^0 \left[\lambda_1^n w_{k,\nu}^{(1)} + \sum_{i=2}^{N} w_{k,\nu}^{(i)} \lambda_i^n \right] \\ &= \sum_{\nu \notin X^*} \sum_{k=1}^{N} \left[p_k^0 w_{k,\nu}^{(1)} + p_k^0 \sum_{i=2}^{N} w_{k,\nu}^{(i)} \lambda_i^n \right] \\ &= \sum_{\nu \notin X^*} \sum_{k=1}^{N} p_k^0 p_\nu^\infty + \left[\sum_{\nu \notin X^*} \sum_{k=1}^{N} p_k^0 \textstyle\sum_{i=2}^{N} w_{k,\nu}^{(i)} \right] |\lambda_2|^n \\ &\leq \sum_{\nu \notin X^*} \sum_{k=1}^{N} p_k^0 p_\nu^\infty + C \cdot |\lambda_2|^n \\ &= \sum_{\nu \notin X^*} p_\nu^\infty \sum_{k=1}^{N} p_k^0 + C \cdot |\lambda_2|^n \\ &= \sum_{\nu \notin X^*} p_\nu^\infty \cdot 1 + C \cdot |\lambda_2|^n \\ &= C \cdot |\lambda_2|^n, \end{aligned}$$

where $C > 0$ and $p_\nu^\infty = 0$ for $\nu \notin X^*$(Lemma 1). Using the complementary probability the proof is completed.

q.e.d.

As a result, the convergence rate of the corresponding Markov chain is mainly determined by the second largest eigenvalue of the diagonalizable transition matrix.

4.2 Matrices in Jordan normal form

In the previous subsection we used diagonalizable matrices for our proof, because they have properties that

can be used advantageously. However, in general the fundamental assumptions for the class of diagonalizable matrices are very restrictive. An arbitrary matrix is often not diagonalizable. Hence, we analyze in this subsection the more general class of matrices in Jordan normal form[1] and illustrate how the proof from the previous subsection can be transferred to matrices in Jordan normal form.

The transition matrix P in Jordan normal form describing the Markov chain of a global convergent GA can be written as (see F. Gantmacher (5))

$$P = \begin{pmatrix} 1 & & & 0 \\ & J_2 & & \\ & & \ddots & \\ 0 & & & J_m \end{pmatrix},$$

where the J_i are called Jordan Blocks with the following shape

$$J_i = \begin{pmatrix} \lambda_i & & & 0 \\ 1 & \lambda_i & & \\ & \ddots & \ddots & \\ 0 & & 1 & \lambda_i \end{pmatrix}.$$

The λ_i are the eigenvalues of P (compare M. Iosifescu, p. 50-51 (10)). The Jordan Blocks have the important property

$$J_i = \lambda_i \cdot E + \underbrace{\begin{pmatrix} 0 & & & 0 \\ 1 & 0 & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{pmatrix}}_{\mathcal{U}},$$

where matrix $\mathcal{U}$ is called nilpotent. A non-negative square matrix $\mathcal{U}$ is defined to be *nilpotent* , if $\exists k \in \mathbb{N}$ holds $\mathcal{U}^k = 0$.

$$P^n = \begin{pmatrix} 1 & & & 0 \\ & J_2 & & \\ & & \ddots & \\ 0 & & & J_m \end{pmatrix}^n =$$

$$= \begin{pmatrix} 1 & & & 0 \\ & J_2^n & & \\ & & \ddots & \\ 0 & & & J_m^n \end{pmatrix}$$

reveals that the unit eigenvalue does not affect the convergence rate. Therefore, J_i^n , $i = 2 \ldots m$, are the remaining parameters that have to be analyzed. With using

$$\mathcal{U} = \begin{pmatrix} 0 & & & 0 \\ 1 & 0 & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{pmatrix}$$

the Jordan blocks become

$$\begin{aligned} J_i^n &= [\lambda_i \cdot E + \mathcal{U}]^n \\ &= \lambda_i^n \cdot E + n\lambda_i^{n-1} \cdot E \cdot \mathcal{U} \\ &+ \ldots + n\lambda_i \cdot E \cdot \mathcal{U}^{n-1} + \mathcal{U}^n. \end{aligned}$$

As mentioned before, matrix $\mathcal{U}$ is nilpotent. This yields

$$\begin{aligned} J_i^n &= \lambda_i^n \cdot E + n\lambda_i^{n-1} \cdot E \cdot \mathcal{U} \\ &+ \ldots + \frac{n(n-1)\ldots(n-k)}{(k-1)!} \lambda_i^{n-k-1} \cdot E \cdot \mathcal{U}^{k-1} \\ &\leq C \cdot \lambda_i^n \cdot E \qquad \text{because } |\lambda_i| < 1, \end{aligned}$$

for $C > 0$ and $k \in \mathbb{N}$. Hence we obtain the final result:

$$J_i^n \leq C \cdot \lambda_i^n \cdot E$$

for $C > 0$ and $k \in \mathbb{N}$.

Interpreting the results, reveals that the J_i^n are upper bounded by λ_i^n for $i = 2 \ldots m$, where λ_i is the corresponding eigenvalue to J_i. The behavior of P^n mainly depends on the size of the second largest eigenvalue λ_2, because in comparison to λ_2^n the powers of the other eigenvalues can be neglected.

The obtained result extends the proof for diagonalizable transition matrices and confirms that the second largest eigenvalue is also the important parameter for the long term behavior of matrices in Jordan normal form.

Finally, we want to note that in general the transition matrix of a Markov chain is not in Jordan normal form, but using standard matrix transformations it always can be transformed into it. Although, we assume that the transformation of an arbitrary transition matrix into a matrix in Jordan normal form does not modify the statements about convergence rate, the formal proof for this is still open.

5 Conclusion

After a short review of two different approaches to the analysis of the convergence behavior of genetic algorithms (building block oriented versus Markov chain

[1] All matrices above $\mathbb{C}$ have a Jordan normal form.

models) we present some requisites we need for our investigation into the convergence rate of genetic algorithms. In section 3 we present some principles of Markov chain models, and review some properties of stochastic matrices. As an investigation into the convergence rate of genetic algorithms only makes sense, if it can be proven that a GA converges to the global optimum, we review the convergence proof from G. Rudolph (13). Using this proof and some of the work by J. Suzuki (15) we can extend the existing convergence models and prove in the following section that the convergence rate of genetic algorithms, modeled with Markov chains is determined by the second largest eigenvalue of the characteristic transition matrix of the Markov chain. Finally, we transfer the results we get for diagonalizable matrices to matrices in Jordan normal form.

This paper extends existing models about convergence rate of GAs (15) and proves that the convergence rate depends on the second largest eigenvalue of the diagonalizable transition matrix. The theoretical analysis of the convergence rate is based on the existing evidence of global convergence, which are based on Markov chains. The obtained results allow a more detailed and deeper theoretical understanding of the principles of evolutionary search. We hope that the results could inspire researchers to put the focus of research more on the underlying theoretical principles and not to focus only on practical applications of GAs.

As a straightforward extension of this work we want to encourage researchers to work on the estimation of the second largest eigenvalue of the transition matrix. A proper approximation can give us information about the optimal choice of GA parameters like mutation and crossover probability or selection pressure. With that knowledge we could use a GA more efficiently, and we would be one step ahead on our long road to the development of competent GAs that are able to solve problems of bounded complexity autonomously, fast and reliably.

References

[1] A. Agapie. Modelling genetic algorithms: From Markov chains to depend ence with complete connections. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefe l, editors, *Parallel Problem Solving from Nature, PPSN V*, pages 3–12, Berlin, 1998. Springer-Verlag.

[2] H. Asoh and H. Mühlenbein. On the mean convergence time of evolutionary algorithms without selection and mutation. *Parallel Problem Solving from Nature- PPSN III*, pages 88–97, 1994.

[3] T. E. Davis and J. C. Principe. A markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, 1(3):269–288, 1993.

[4] A. E. Eiben, E. H. L. Aarts, and K. M. Van Hee. Global convergence of genetic algorithms: An infinite Markov chain Analysis. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 4–12, Berlin, 1991. Springer-Verlag.

[5] F. Gantmacher. *Matrizentheorie*. Berlin, Springer Verlag, 1986.

[6] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Reading Addison-Wesley, 1989.

[7] D. E. Goldberg. The race, the hurdle, and the sweet spot. In P. J. Bentley, editor, *Evolutionary Design by Computers*, pages 105–118. Morgan Kaufmann, San Francisco, CA, 1999.

[8] D. E. Goldberg and P. Segrest. Finite Markov chain analysis of genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms*, pages 1–8, 1987.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Cambridge, Massachusetts, 2. edition, 1998.

[10] M. Iosifescu. *Finite Markov Processes and their applications*. Chichester, John Wisly and Sons, 1980.

[11] A. E. Nix and M. D. Vose. Modelling genetic algorithms with markov chains. *Annals of Mathematics and Artifical Intellignece*, 2:79–88, 1992.

[12] V. Romanovsky. *Discrete Markov chains*. Wolters-Noordhoff Publishing, Groningen, 1978.

[13] G. Rudolph. *Convergence properties of evolutionary algorithms*. Kovač, Hamburg, 1997.

[14] W. J. Stewart. *Introduction to the Numerical Solution of Markov chains*. Princeton University Press, Princeton, New Jersey, 1994.

[15] J. Suzuki. A markov chain analysis on simple genetic algorithms. *IEE Transactions on Systems, Man and Cybernetics*, 25(4):655–659, 1995.

[16] D. Thierens. *Analysis and design of genetic algorithms*. Katholieke Universiteit Leuven, Leuven, Belgium, 1995.

[17] D. Thierens and D. E. Goldberg. Convergence models of genetic algorithm selection schemes. *Parallel Problem Solving from Nature- PPSN III*, pages 119–129, 1994.

[18] M. D. Vose and G. E. Liepens. Punctuated equilibria in genetic algorithms. *Complex Systems*, 5:31–44, 1991.

The No Free Lunch and Problem Description Length

C. Schumacher
Department of Computer Science
University of Tennessee, Knoxville
Knoxville, Tennessee 37996 USA
schumach@cs.utk.edu

M. D. Vose and L. D. Whitley
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523 USA
{vose,whitley}@cs.colostate.edu

Abstract

The No Free Lunch theorem is reviewed and cast within a simple framework for blackbox search. A duality result which relates functions being optimized to algorithms optimizing them is obtained and is used to sharpen the No Free Lunch theorem. Observations are made concerning problem description length within the context provided by the results of this paper. It is seen that No Free Lunch results are independent from whether or not the set of functions (over which a No Free Lunch result holds) is compressible.

1 Introduction

Roughly put, the No Free Lunch theorem formalizes the intuitive idea that all blackbox search algorithms have identical behavior over the set of all possible discrete functions. Thus, on average, no algorithm is better than random enumeration in locating a global optimum. If algorithms are executed any given number of steps, every algorithm finds the same set of best so-far solutions over all functions [9] [5] [1].

One of the criticisms of the No Free Lunch theorem is that it applies to large sets of functions and it is unclear if No Free Lunch applies to small sets or to real world problems of practical interest. A variant form of this criticism is that many practical problem classes have compact descriptions, whereas elements in the set of all functions from a finite domain to a finite codomain do not have (on average) compact descriptions. This criticism has previously been addressed by various researchers [5] [2], where it was observed that a No Free Lunch result holds over classes of functions much smaller than the set of all functions. This paper strengthens those observations, obtaining a sharpened version of the No Free Lunch theorem, and also makes more explicit a type of duality involving functions being optimized and algorithms being used to optimize them. The paper closes with observations regarding the No Free Lunch theorem and problem description length.

2 Search Algorithm Framework

This section sets forth a framework for the analysis of deterministic non-repeating blackbox search algorithms. To streamline exposition, such search algorithms will be referred to simply as algorithms. This framework makes it possible to precisely model all possible algorithms as they apply to all functions of a given finite domain and range.

2.1 Definitions

Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets, let $f : \mathcal{X} \to \mathcal{Y}$ be a function, and define y_i as $f(x_i)$. Define a *trace* of size m ($m \geq 0$) to be a sequence of pairs

$$T_m \equiv \langle (x_0, y_0), (x_1, y_1), \ldots, (x_{m-1}, y_{m-1}) \rangle$$

Note that a trace is just an ordered sequence of elements from f (regarding f as a set of ordered pairs). At times the subscript of a trace will be omitted to refer to traces of arbitrary size. Let $\mathcal{T}_m$ be the set of all traces of size m, and let $\mathcal{T}$ be the set of all traces. Adopt the following notation:

$$\begin{aligned} T_0 &= \langle\rangle \\ T_m^x &\equiv \langle x_0, x_1, \ldots, x_{m-1} \rangle \\ T_m^y &\equiv \langle y_0, y_1, \ldots, y_{m-1} \rangle \\ T_m[i] &\equiv (x_i, y_i) \\ T_m^x[i] &\equiv x_i \\ T_m^y[i] &\equiv y_i \end{aligned}$$

A concatenation operator $\|$ will be used to extend the size of a trace in the following way:

$$T_m \parallel (x, y) \quad \equiv \quad \langle T_m[0], T_m[1], \ldots, T_m[m-1], (x, y) \rangle$$

Define a *non-repeating trace* T to be a trace with unique x components, i.e. $T^x[i] = T^x[j] \Rightarrow i = j$.[1] A *complete trace* T is defined to be a trace that covers the domain, i.e. for all $z \in \mathcal{X}$ there exists an i such that $T^x[i] = z$. Because a trace is a sequence of ordered pairs, a non-repeating trace corresponds to a function; when it is complete, the corresponding function is f.

Consider a "search" operator $g : \mathcal{T} \to \mathcal{X}$ which when given a trace as an argument returns the next point in the search space to be examined. A *deterministic blackbox search algorithm* A corresponds to a search operator g, and takes as arguments a trace T_m and a function $f \in \mathcal{Y}^{\mathcal{X}}$ and returns the trace

$$T_{m+1} = A_f(T_m) = T_m \parallel (g(T_m), f \circ g(T_m))$$

For example, the first two steps of deterministic blackbox search algorithm A would proceed as follows:

$$\begin{aligned} T_1 = A_f(T_0) &= T_0 \parallel (g(T_0), f \circ g(T_0)) \\ T_2 = A_f(T_1) &= T_1 \parallel (g(T_1), f \circ g(T_1)) \end{aligned}$$

Such algorithms therefore operate in discrete steps where each step generates a new pair that is concatenated into the trace. Note that the search operator g is used to generate the x components of the trace, and that function f is used to evaluate the utility of those points; this reflects the separation between "exploration" (choosing the next point in the search space) and "fitness evaluation" (evaluating the utility of that new point). Multiple applications of these algorithms will be abbreviated in the natural way, i.e. $A_f^m(T_0) = T_m$, and in particular, $A_f^0(T_0) = T_0$.

A *non-repeating blackbox search algorithm*—referred to simply as algorithm—is defined to be a blackbox search algorithm whose range contains only non-repeating traces. The largest trace an algorithm could generate is clearly a complete trace which has size $|\mathcal{X}|$.

After m steps, algorithm A and function f will generate trace T_m from initial trace T_0. In this paper algorithms always start from the empty trace T_0, which may seem a limitation. However, algorithms with an arbitrary initial trace size are actually special cases of algorithms that start from the empty trace, as the following illustrates: Consider algorithm A and initial trace T_m. A corresponds to another algorithm A' that given initial trace T_0 will generate T_m after m steps, and will behave exactly as A afterwards. Designating an initial trace is thus simulated by using a slightly modified algorithm that starts at T_0. In other words, algorithms that can set all points in their traces are powerful enough to encompass algorithms that cannot.

Two algorithms A and B will be considered identical if and only if they both generate the same complete trace for all $f \in \mathcal{Y}^{\mathcal{X}}$, i.e.:

$$A_f^{|\mathcal{X}|} = B_f^{|\mathcal{X}|} \text{ for all } f \in \mathcal{Y}^{\mathcal{X}}.$$

2.2 No Free Lunch

Define a *performance vector* of length m to be a sequence of m values from $\mathcal{Y}$. The performance vector associated with trace T_m is T_m^y. A performance vector can thus be said to be *derived* from a trace, and a function and an algorithm together can be said to generate a performance vector from T_0.

The length m trace $A_f^m(T_0)$ generated by algorithm A and function f will be abbreviated by $T_m(A, f)$. Let $V_m(A, f)$ denote the length m performance vector generated by A and f. The size subscripts may be omitted when not needed. Note that the performance vector $V_m(A, f)$ is closely related to $T_m(A, f)$,

$$V_m(A, f) = (T_m(A, f))^y$$

Define an *overall measure* of algorithm A and set of functions F to be a function that maps the set of performance vectors generated by A and F to a real number. An overall measure can be used to compare the overall performance of two algorithms on a set of functions, and if the two algorithms have identical overall measures, it can be said that they perform equally well over F. An example of an overall measure would be to take a performance vector measure M (which maps a performance vector to a real number), apply it to every element in F and then combine the results in some symmetric way, such as the average $\sum_{f \in F} M(V(A, f))/|F|$.

Define an *No Free Lunch result over* F to be a situation where any two algorithms will have equal overall performance with respect to the set of functions F. Four equivalent statements of the No Free Lunch theorem are given below. Where ambiguous, the set of functions involved is $\mathcal{Y}^{\mathcal{X}}$.

NFL1: For any overall measure, each algorithm performs equally well.

The following is the pivotal idea contained in the proof

[1] This paper will follow the convention that free variables are universally quantified.

by Radcliffe and Surry [5], phrased more directly in the language of the current framework.

NFL2: For any two algorithms A and B, and for any function f, there exists a function g such that $V(A,f) = V(B,g)$.

The following is the basis of the No Free Lunch proof given by Schumacher[6].

NFL3: Every algorithm generates precisely the same collection of performance vectors when all functions are considered.

Schumacher has proved in addition that $V(A,f) = V(A,g) \Longrightarrow f = g$, i.e., the collections referred to in NFL3 are actually sets [6]. This fact is a key observation in demonstrating the equivalence of NFL1, NFL2, NFL3, and NFL4.

As defined above, an overall measure is a function of a set of performance vectors. Consider instead a *weighted overall measure* in which a performance vector measure M applied to each performance vector is weighted according to the function that generates it, i.e. $W(f)M(V(A,f))$, and summed over f. A weighted overall measure is *not* generally subject to the No Free Lunch theorem except in the case where the functions are *equally weighted*, i.e. certain functions are not deemed more important than others. The statement below is essentially the No Free Lunch result given in Wolpert and Macready [8].

NFL4: For any equally weighted overall measure, each algorithm will perform equally well.

A corollary of the No Free Lunch theorem is that if an algorithm performs better than average on one set of functions, it must perform worse on the complementary set. This is essentially an argument for specialization: an algorithm will perform well on a small set of functions at the expense of poor performance on the complementary set.

An even stronger consequence which seems not to have been properly appreciated is that *all* algorithms are equally specialized. This contradicts commonly stated beliefs (e.g. [4]) about how there can be *robust* general purpose algorithms, meaning that they perform reasonably well on a broad class of functions at the expense of not performing extremely well on any set of functions. Since every algorithm has precisely the same collection of performance vectors when all functions are considered (NFL3), it follows that *if any algorithm is robust, then every algorithm is, and if some algorithm is not robust, then no algorithm can be!*

3 Sharpening No Free Lunch

Let $f : \mathcal{X} \to \mathcal{Y}$ be a function and let $\sigma : \mathcal{X} \to \mathcal{X}$ be a permutation (i.e. σ is one-to-one and onto). The permutation σf of f is the function $\sigma f : \mathcal{X} \to \mathcal{Y}$ defined by $\sigma f(x) = f(\sigma^{-1}(x))$.

Define a set F of functions to be *closed under permutation* if for every $f \in F$, every permutation of f is also in F.

Let A be an algorithm with search operator g and let σ be a permutation (of $\mathcal{X}$). The permutation σA of A is the algorithm with search operator σg defined by $\sigma g(\phi) = \sigma^{-1}(g(\sigma_x(\phi)))$ where $\sigma_x(\phi)$ operates on the x values of trace ϕ by applying σ to each of them, while leaving the y values untouched.

THEOREM: If

$$T_n(A, \sigma f) = \langle (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \rangle$$

then

$$T_n(\sigma A, f) = \langle (\sigma^{-1}(x_0), y_0), \ldots, (\sigma^{-1}(x_{n-1}), y_{n-1}) \rangle$$

Proof: By induction on the length of the traces. The base case is true since all traces of length 0 are the same; $T_0(\sigma A, f) = T_0(A, \sigma f) = \langle\rangle$. Assume the inductive hypothesis (i.e., the equalities in the statement of the theorem). By definition,

$$\begin{aligned} \sigma g(T_n(\sigma A, f)) &= \sigma^{-1} \circ g(\sigma_x(T_n(\sigma A, f)) \\ &= \sigma^{-1} \circ g(T_n(A, \sigma f)) = \sigma^{-1}(x_n) \end{aligned}$$

Moreover, $f(\sigma^{-1}(x_n)) = \sigma f(x_n) = y_n$. Accordingly:

$$\begin{aligned} T_{n+1}(A, \sigma f) &= T_n(A, \sigma f) \mid\mid (x_n, y_n) \\ T_{n+1}(\sigma A, f) &= T_n(\sigma A, f) \mid\mid (\sigma^{-1}(x_n), y_n) \end{aligned}$$

Which completes the proof. □

COROLLARY ("Duality"): $V(\sigma A, f) = V(A, \sigma f)$

This Corollary is true since, by the previous theorem, the y values are the same in both traces. This corollary is striking in the way that it shows a correspondence between a permutation of an algorithm and a permutation of a function. The following Lemma is an easy consequence of NFL2.

LEMMA1: If the set of functions F is closed under permutation, then there is a No Free Lunch result over F.

Proof: Let A and B be arbitrary algorithms. If one can show the sets $S_1 = \{V(A,f) : f \in F\}$ and $S_2 = \{V(B,h) : h \in F\}$, are equal, then any two algorithms will provide the same data for computing

their combined performance measures, and therefore the same result will be obtained. By NFL2, there exists a function h such that $V(A,f) = V(B,h)$. Because these two performance vectors are equal, h must be a permutation of f, and thus $f \in F \Longrightarrow h \in F$. Hence $S_1 \subset S_2$. The reverse containment follows by symmetry. □

The previous lemma was an intermediate result in Radcliffe and Surry's proof of the No Free Lunch theorem [5]. The converse of this lemma is also true.

LEMMA2: If a No Free Lunch result holds over the set of functions F, then F is closed under permutation.

Proof: Assume by way of contradiction that a No Free Lunch result holds over the set F, but that F is not closed under permutation, i.e., the function $f \in F$ has a permutation g which is not in F. Consider an arbitrary algorithm A. Let $M(V(A,f)) = 1$, and let M equal zero for all other performance vectors generated by A. By NFL3 and the paragraph following it, for every algorithm B there exists a function h_B (the subscript on h indicates dependence on B) such that $M(V(B,k)) = 1 \iff k = h_B$. Let the overall measure be the sum $\sum_{k \in F} M(V(B,k))$. Note that this sum is 1 when $B = A$, and since a No Free Lunch result is assumed over F, the sum is 1 for every algorithm B. As f and g are permutations, let $f = \sigma g$. By duality, $V(A,f) = V(A,\sigma g) = V(\sigma A, g)$, and thus $M(V(\sigma A, g)) = 1$. Accordingly, $\sum_{k \in F} M(V(\sigma A, k)) = 0$ (since $M(V(\sigma A, k))$ is non zero only for $k = h_{\sigma A} = g \notin F$), a contradiction. □

Combining the previous lemmas yields the following sharpened version of the No Free Lunch theorem:

NFL: A No Free Lunch result holds over the set of functions F if and only if F is closed under permutation.

4 NFL and Permutation Closure

In this section, some consequences of the previous results are illustrated.

Define the permutation closure $P(F)$ of a set of functions $F \subset \mathcal{Y}^{\mathcal{X}}$ by

$$P(F) = \{\sigma f : f \in F, \text{ and } \sigma \text{ is a permutation (of } \mathcal{X})\}$$

Note that for any sets F, F' of functions (from $\mathcal{Y}^{\mathcal{X}}$),

$$P(F \cup F') = P(F) \cup P(F')$$

By construction, $P(F)$ is closed under permutation and therefore a No Free Lunch result holds over $P(F)$ for any set $F \subset \mathcal{Y}^{\mathcal{X}}$ (and hence over unions of such sets). It bears mentioning that in particular NFL1, NFL2, NFL3, and NFL4 are valid with respect to $P(F)$. Not only do all algorithms display equal behavior over $P(F)$ for some overall measure of performance (NFL1), they also generate exactly the same set of performance vectors (NFL3) and therefore have identical collections of objective function values at every time step.

An equivalence relation $\equiv$ may be defined with respect to permutations. Functions f and g are said to be equivalent, denoted by $f \equiv g$ if and only if there exists a permutation σ (of $\mathcal{X}$) for which $f = \sigma g$. Similarly, algorithms A and B are said to be equivalent, denoted by $A \equiv B$ if and only if there exists a permutation σ (of $\mathcal{X}$) for which $A = \sigma B$.

Let the equivalence class of function f be denoted by $[f]$, and let the equivalence class of algorithm A be denoted by $[A]$. To simplify notation, let $\mathcal{A}$ denote a set of algorithms, let $\mathcal{F}$ denote a set of functions, and define $V(A,\mathcal{F})$ and $V(\mathcal{A},f)$ as follows

$$\begin{aligned} V(A,\mathcal{F}) &= \{V(A,f) : f \in \mathcal{F}\} \\ V(\mathcal{A},f) &= \{V(A,f) : A \in \mathcal{A}\} \end{aligned}$$

Since $[f] = P(\{f\})$, NFL applies; therefore, for any given algorithms A and B,

$$V(A,[f]) = V(B,[f])$$

It follows immediately from the definitions that if F is closed under permutation and $f \in F$ then $[f] \subset F$. Therefore the case above (i.e., $F = [f]$) is the finest level of granularity at which a No Free Lunch result can hold. Moreover, any set F of functions closed under permutation is a disjoint union of equivalence classes, thus No Free Lunch results hold only over unions of equivalence classes.

By definition and duality,

$$\begin{aligned} V([A],f) &= \{V(\sigma A,f) : \sigma \text{ is a permutation}\} \\ &= \{V(A,\sigma f) : \sigma \text{ is a permutation}\} \\ &= V(A,[f]) \end{aligned}$$

Bringing NFL into the picture yields the result that for any given algorithms A and B,

$$V([A],f) = V(A,[f]) = V(B,[f]) = V([B],f)$$

It follows that for any given algorithm A and any given function f, the following are identical:

- The average performance over all algorithms using function f.

- The average performance over an arbitrary equivalence class of algorithms using function f.
- The average performance over all functions in the equivalence class $[f]$ using algorithm A.

Moreover, the phrase "average performance" can be replaced with "set of performance vectors" in the list above. Whereas most No Free Lunch results have been expressed in terms of some measure of performance, all algorithms in fact display exactly the same behavior over any set of functions closed under permutation in the sense that the performance vectors are identical.

5 NFL Equivalence Class Examples

In this section, some extreme examples of permutation closure are presented. These examples not only illustrate applications of NFL, but also set the stage for discussing the notion of problem description length.

For conciseness, a function will be represented by a list of its output values (i.e., as a sequence; the points of the domain are implicitly the indices into the sequence).

The smallest permutation closures correspond to functions that return a single value. For example,

$$f = \langle 0,0,0,0 \rangle \implies [f] = \{f\}$$

Such problems are in some sense uninteresting from a search point of view, since a single evaluation automatically determines the maximum and minimum of the evaluation function.

The smallest sets corresponding to a permutation closure where the evaluation functions display variability are needle-in-a-haystack functions. Such a function f has the same evaluation (call it 0) everywhere except at one point in the domain, where a better evaluation is found (call it 1). Since there is exactly one point in the space with a different evaluation, the size of $[f]$ is $|\mathcal{X}|$. For example

$$\begin{aligned} f &= \langle 0,0,0,1 \rangle \implies \\ [f] &= \{\langle 0,0,0,1\rangle, \langle 0,0,1,0\rangle, \langle 0,1,0,0\rangle, \langle 1,0,0,0\rangle\} \end{aligned}$$

An interesting class of functions is the set of decision problems which return Boolean values ($\mathcal{Y} = \{0,1\}$). Note that NP-Complete problems are frequently defined as particular decision problems. This class is in one-to-one correspondence with the set of length $N = |\mathcal{X}|$ binary strings, and is therefore equal to its permutation closure. It is moreover a disjoint union of equivalence classes $[f_1], \ldots, [f_N]$ where

$$\begin{aligned} f_i &= \langle \underbrace{1 \ldots 1}_{i \text{ times}}, 0, \ldots, 0 \rangle \\ |[f_i]| &= \binom{N}{i} \end{aligned}$$

As a final example, consider functions that are one-to-one and onto. This class of functions also equals its permutation closure. Without loss of generality, $\mathcal{X} = \mathcal{Y}$ and such functions are permutations. There is a single equivalence class, namely $[I]$ where I is the identity function, and its size is $N!$ (where $N = |\mathcal{X}|$).

6 Problem Description Length

The average description length for functions that are members of a permutation closure is discussed in this section. Although this is only done for select cases, the cases illustrate the extremes in average description length.

Whitley [7] has previously made (a variation on) the observation that given any permutation σ (on $\mathcal{X}$), the permutation closure $[\sigma]$ is the set of all N! permutations, and the average description length for its members is $\Omega(N \ln N)$ bits, where $N = |\mathcal{X}|$.

A more general observation is that given any set F of functions, the average description length of members in $P(F)$ is $\Omega(\ln k)$ bits, where $k = |P(F)|$.

An interesting question is: when is the the average description length over the members of some permutation closure polynomial and when it it exponential? A correct, but somewhat circular answer, is that the average description length is polynomial when $\ln k$ is polynomial. Nevertheless, we can still use this idea to examine average description length for examples which provide bounding cases.

It has already been noted above that the average description length for permutations is $\Omega(N \ln N)$ bits. Note, moreover, that an explicit definition of a permutation (as a sequence, as described in the previous section) would take $O(N \ln N)$ bits; there are N images (positions in the sequence) to define, and each takes $O(\ln N)$ bits (since there are N points in the range). Therefore, on average, the permutation closure $[\sigma]$ contains *incompressible functions*; the average description length of a member is the same order of magnitude as the size of an explicit definition (as a sequence).

At the other extreme is the permutation closure $[f]$ of a needle-in-a-haystack function f (described above).

Explicit definition of a member (of $[f]$) requires $\Omega(N)$ bits, whereas the average description length is $O(\ln N)$ bits. Therefore members of this permutation closure are highly compressible.

These two extreme cases illustrate that No Free Lunch results are independent from whether or not the set of functions (over which a No Free Lunch result holds) is compressible.

7 Conclusions

No Free Lunch theorems in various equivalent forms are reviewed. A duality result is proven and used to obtain a sharpened No Free Lunch theorem, in the sense that both necessary and sufficient conditions are obtained.

It is proven that the permutation closure of a single function is the finest level of granularity at which a No Free Lunch result can hold. The average description length of members of permutation closures is computed (for select cases) and is related to compressibility. It is seen that No Free Lunch results are independent from whether or not the set of functions (over which a No Free Lunch result holds) is compressible.

References

[1] J. Culberson. On the Futility of Blind Search. *Evolutionary Computation*, 6(2):109–127, 1999.

[2] T. English. Practical implications of new results in conversation of optimizer performance. In Schoenauer et al., editor, *Parallel Problem Solving from Nature, 6*, pages 69–78, 2000.

[3] Thomas English. Information is Conserved in Optimization. *IEEE Trans Evolutionary Computation.*

[4] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[5] N.J. Radcliffe and P.D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In J. van Leeuwen, editor, *Lecture Notes in Computer Science 1000.* Springer-Verlag, 1995.

[6] C. Schumacher. *Fundamental Limitations of Search.* PhD thesis, University of Tennessee, Department of Computer Sciences, Knoxville, TN, 2000.

[7] D. Whitley. Functions as permutations: regarding no free lunch, walsh analysis and summary statistics. In Schoenauer et al., editor, *Parallel Problem Solving from Nature, 6*, pages 169–178, 2000.

[8] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, July 1995.

[9] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82, 1997.

A Building Block Favoring Reordering Method for Gene Positions in Genetic Algorithms

Onur Tolga Sehitoglu
Dept. of Computer Engineering
Middle East Technical University, Ankara
onur@ceng.metu.edu.tr

Göktürk Üçoluk
Dept. of Computer Engineering
Middle East Technical University, Ankara
ucoluk@ceng.metu.edu.tr

Abstract

This work proposes an algorithm to speed-up convergence of genetic algorithms that is based on the investigation of neighbouring gene values of the successful individuals of the chromosome pool. By performing some statistical inference on neighbour gene values, coherent behaving genes are detected. It is claimed that those coherent acting genes are belonging to the same building block that leads to the solution. It is desirable to keep genes of the same building block together to let them —probabilistically— live together in next generations. Therefore, a reordering of the gene positions is performed, periodically. The proposed algorithm is implemented for a minimum area non-overlapping rectangle placement problem and results are compared to a classical GA implementation.

1 Introduction

n-point crossover is known to favor the formation of the so called *building blocks*. Building blocks are those relatively short gene subsequences which, when combined, form better solutions. The mechanism of genetic algorithm (GA) implicitly assigns higher chances to building blocks to exist over generations. In other words, to survive.

The idea in this work is based on the fact that building blocks mainly consist of those group of genes which are closely located to each other. Now, consider two genes, which due to the nature of the problem would tend to form a building block, and are separated by some other genes in the chromosome in the default gene ordering of the chromosome representation. These two will not easily be able to form a building block, when subject to n-point crossover; because the probability of hitting some intermediate point as the cut-point of the crossover is high. Therefore the order of genes in the chromosome encoding plays an important role in the convergence of the GA.

We propose a method in which permutations of the gene ordering are considered dynamically. In Section 2 the proposed method is described. The method is implemented for a minimum non-overlapping rectangle placement problem and the implementation is described in Section 3. In Section 4 results are compared with the classical GA's results.

2 Proposed Method

In each generation a single global permutation is considered. This permutation maps a gene number to a position value in the chromosome encoding. The crossover operation is based on this mapping instead of the original gene order in the representation. During the evolution of the genetic algorithm, this global permutation is readjusted by means of statistical analysis on neighbouring genes, calculated for the bests of the pool.

- Consider that the solution to a subject problem requires the determination of a set of parameters $\mathcal{X}$. As the first step GA requires the determination of a one-to-one mapping from the set of parameters to the set of binary strings $\mathbf{\Gamma}$, this is called the *encoding* of the parameters.

  $$\mathcal{E}_i : \mathcal{X}_i \mapsto \Gamma_i \quad \text{where} \quad \mathcal{X}_i \in \mathcal{X}, \ \ \Gamma_i \in \mathbf{\Gamma}$$

  We name Γ_i as *genes*. Γ_i's, each of which are binary strings, are concatenated into a one dimensional binary array $\mathbf{\Gamma}$ so that Γ_i is followed by Γ_{i+1}. An instance of $\mathbf{\Gamma}$ is called a *chromosome*.

- We define a *permutation of genes* as an ordering

relation of the genes in a chromosome. If $\mathcal{P}_p$ represents such a permutation operator on a chromosome, it is defined by means of its gene elements as:

$$[\mathcal{P}_p\Gamma]_i \triangleq \Gamma_{p(i)}$$

Here p is a permutation function:

$$p : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, n\} \quad \text{where}$$

$$n = |\mathbf{\Gamma}| \quad \text{and} \quad p^{-1} \quad \text{exists.}$$

- As a part of the proposed method we define for each gene position i of the chromosome a function f_i that admits a gene value as arguments and is defined as:

  $$F_i : \Gamma_i \mapsto \mathcal{R} \quad \text{where} \quad \Gamma_i \in \mathbf{\Gamma}$$

  Please also note that, in the most general case, Γ_i can be a tuple representation of various features. Therefore F_i may be defined over the corresponding allele tuple domain.

  For denotational simplicity, we will define

  $$f_i \triangleq F_i(\Gamma_i)$$

  so f_i is a real value which is calculated from a Γ_i by means of F_i.

  When the *building blocks* of a GA is reverse mapped to the actual problem domain, it is usually observed that the formation of blocks corresponds to some patternal, structural, mathematical invariance or covariances.

  The functions F_i will serve to express features as real values which will be used to discover some invariance or covariances. So, in a way, we are defining a handle, where the GA user has a chance to hook-in his hint for defining the features which may lead to building blocks.

- At each generation we calculate a *neighbour-affinity-function* $\mathcal{A}_i$ that is defined for each neighbour gene pair position in the current permutation over the whole pool. We propose it to be of the most general form:

  $$\mathcal{A}_i^p : \{\langle f_{p^{-1}(i)}, f_{p^{-1}(i+1)}\rangle\}_{pool} \mapsto [0, 1] \quad \text{where}$$

  $$i = 1, 2, \ldots, n-1$$

  Of course if the chromosome composition is a vector of genes (due to the nature of the problem) then all f_i's will reduce to a single f and hence $\mathcal{A}_i$ will reduce to a single function $\mathcal{A}$.

  $\mathcal{A}_i$ is a problem specific function and should be coded according to the characteristics of the problem encoding. Since the aim is to construct good building blocks, the result of $\mathcal{A}_i^p$ should be closer to 1 for gene values acting coherently and contributing for better solutions. Correlation and standard deviation analysis can be used as alternatives for $\mathcal{A}_i$.

- The next step is to modify the permutation mapping by looking at the results of neighbour-affinity value calculations which are calculated from the instances of the current generation of the pool. We will be calling these values A_i^p.

  To do this, every gene position in the permutation will be considered, and based on the right and left neighbour-affinity values of each gene a decision will be made for ots position. If this value is found to be less then a threshold value τ, these two genes will be considered as unrelated to each other and the permutation will be changed to separate them. Actually there exists 4 possible affinity cases for a gene:

  1. *Affinity value is greater then τ for both neighbours* so there is no problem with the current ordering. The gene position is kept in the permutation unaltered.
  2. *Affinity value with left gene is greater than τ but it is less than τ for the right gene.* This means left neighbour is okay but we should separate it from the right. So, gene exchanges position with the left neighbour. In this way the good affinity with the left neighbour is preserved.
  3. *Affinity value with left gene is less than τ but it is greater than τ for the right gene.* Similarly gene exchanges its position with the right neighbour.
  4. *Both affinity values are less than τ.* Gene is moved to a random position in the permutation.

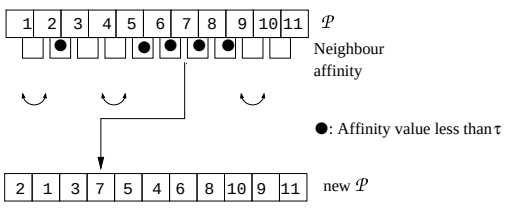


Figure 1: Modification of order according to affinity values

Modifications in the permutation mappings are kept to be as local as possi-

ble. The algorithm to do this is as follows:

for $i \leftarrow 2 \ldots n$ **do**
{
if $p[i-1]$ *is not modified in prev. iter.*
if $A_{i-1} > \tau \wedge A_i < \tau$
$p[i-1] \leftrightarrow p[i]$
else if $A_{i-1} < \tau \wedge A_i > \tau$
$p[i] \leftrightarrow p[i+1]$
else if $A_{i-1} < \tau \wedge A_i < \tau$
Move $p[i]$ *to a random location*
}

2.1 GA Engine

- Initialize the global permutation $\mathcal{P}$ to $[1, 2, \ldots, n]$
- Generate a random population, evaluate it and store it also as the former generation.
  Pool size = 100 chromosomes

Repeat :

- Mutate.
  Mutation Rate = Once each 10 generation one random chromosome
  Changes per Chromosome = Flip one randomly selected gene
- Mate all the pool by forming random pairs.
  Determine the crossover points.
  Perform crossovers among the chromosomes according to the permutation. Crossover point is taken in permutation mapping.
  Cross Over = At 10 random chosen random length gene intervals
- Evaluate the new generation.
  Keep Ratio = At most 10%
- If reorder period is reached:
  - Calculate A_i^p values for the selected-kept chromosomes according to current permutation mapping.
  - Modify the ordering of each gene position according to A_i values and find the new permutation.
- Display/Record performance result.
- If it was not the last generation the user demanded, **goto Repeat**.

3 Implementation

For testing and implementation of the proposed system, a minimum area rectangle placement problem is chosen. In this problem a set of rectangles is given as input. The aim is to find a placement of all rectangles such that all rectangles are in a bounding box which is minimized in the area and no two rectangles intersect. The input is the width and height information of n rectangles.

3.1 Problem encoding

- $I_i = \langle w_i, h_i \rangle$ is the width & height input of the i^{th} rectangle.
- $\Gamma_i = \langle x_i, y_i, o_i \rangle$ where x_i and y_i are offsets from the origin and $o_i \in \{0, 1\}$ is the orientation (not rotated, rotated 90°) in the placement. Each gene Γ_i defines a placement for I_i. Γ_i in combination with I_i describes a rectangle positioning with absolute coordinates.
- The fitness function for a chromosome is defined as:
  $V = cZ + dO, +eB \quad j = 1, \ldots, poolsize$
  where c, d, e are positive constant weights and Z is the total area of the rectangles placed out of the placement area, O is the total overlapping area among all rectangle pairs, B is the area of the minimum bounding box covering all rectangles in the placement. The aim of the genetic algorithm is to find a chromosome that *minimizes* V.
- One point crossover for crossover point k is defined by means of the permutation mapping $\mathcal{P}$ as:

$$\Gamma_i^{AB} = \begin{cases} \Gamma_i^A & \text{if } p(i) < k \\ \Gamma_i^B & \text{if } p(i) \geq k \end{cases}$$

$$\Gamma_i^{BA} = \begin{cases} \Gamma_i^B & \text{if } p(i) < k \\ \Gamma_i^A & \text{if } p(i) \geq k \end{cases}$$

  where Γ^A and Γ^B are crossedover to produce two offsprings: Γ^{AB} and Γ^{BA}.
- Since the gene values are three-tuples of numeric values, there is no need to define an auxilary function f_i which will map them to $[0, 1]$. Their numerical values are directly used in the formula.
- The neighbour affinity function is defined to be the total standart deviation of offset values describing the placement:
  $A_i^p = 1 - (t\sqrt{\sigma_x(i)} + u\sqrt{\sigma_y(i)} + v\sqrt{\sigma_o(i)})$
  where $\sigma_\alpha(i)$ is defined as the variance of the differences of the α features of the $\langle \Gamma_{p^{-1}(i)}, \Gamma_{p^{-1}(i+1)} \rangle$ tuples in the population. t, u, v are positive constant weights:

$$\sigma_\alpha(i) = \frac{\sum_{j=1}^{M} \delta_{i,j,\alpha}^2 - (\sum_{j=1}^{M} \delta_{i,j,\alpha})^2/M}{M}$$

$$\delta_{i,j,\alpha} = \Gamma_{p^{-1}(i),j,\alpha} - \Gamma_{p^{-1}(i+1),j,\alpha}$$

Due to the proposed algorithm, when A_i^p is small then two neighbour genes should be placed in relatively arbitrary positions in the population. This is so, a small A_i^p value means that they are statistically found not to cooperate well towards the solution. When A_i^p values become close to 1, the relative placement of neighbour genes is almost fixed in the population which means a building block is established by these neighbours.

- Reorder frequency is chosen as 5. That means, analysis of A_i values and permutation modification is done once in 5 generations.

3.2 Test results

The resulsts of the implementation of the proposed algorithm to the resulst of the classical GA implementation where all other paramaters like crossover operations, mutation frequency, keep ratio[1] are the same but no permutation shuffling is done. In the implementation of the proposed method, crossovers are carried out by using the permutation information, and permutation mapping is modified once in 5 generation. Tests are repeated 20 times with different random seeds. In Figure 2 the evolution of the best individual fitness value is indicated. The proposed gene reordering method converges significantly faster. It reaches the mimimum in 150 generations compared to 550 to 600 generations of the classical GA version.

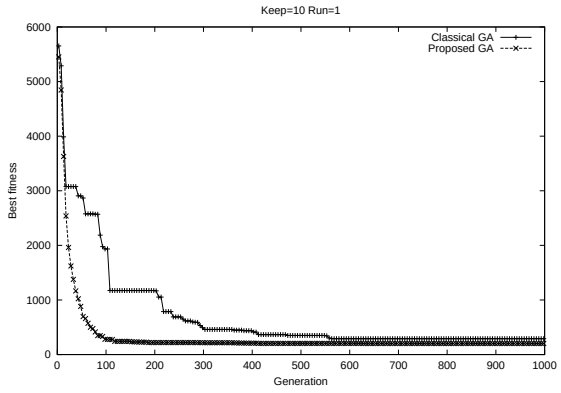


Figure 2: Evolution of the best individual for a sample execution

[1]Ratio of the best members of current generation to be kept in the next generation

Distributions of the best individuals for all 20 execution cases through the generations is given in Figure 3. The gene reordering version is consistently converging to a solution in less number of generations for all cases. furthermore, the classical version exhibits a slow convergence behaviour which is also likely to get trapped into a local minima more frequently.

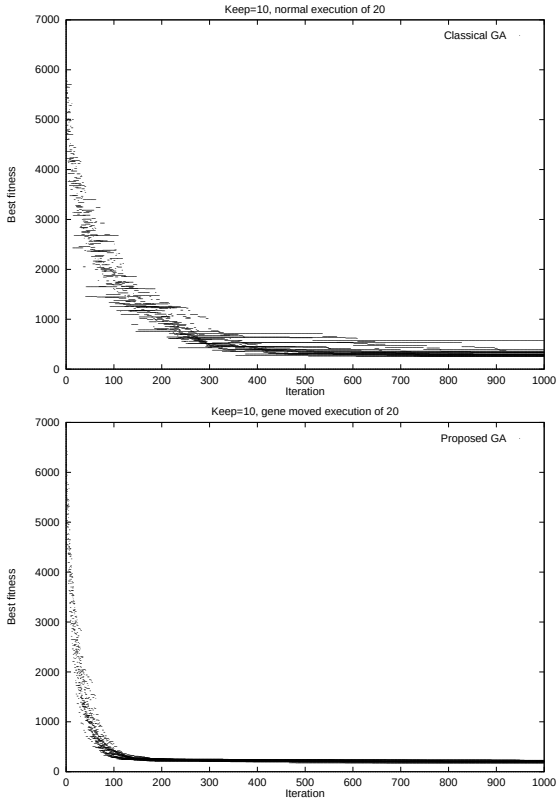


Figure 3: Distribution of the best fitness values for 20 executions where *Keep* ratio is 10

When the fitness values of the best individuals after 1000 iterations are compared it is observed that though the proposed algorithm converges significantly faster it does not find a worse solution than the original algorithm (Figure 4). In contrary, since it gets caught in local minima, the classical algorithm requires more than 1000 generations to achive the quality of the solution which the proposed GA reaches in 150 generations.

Considering the reordering cost, it is also observed that after a small number of generations neighbour affinity values gets under the specified threshold and the system does not require a permutation change. In most of the cases a fixed permutation was converged to in 50 to 100 generations.

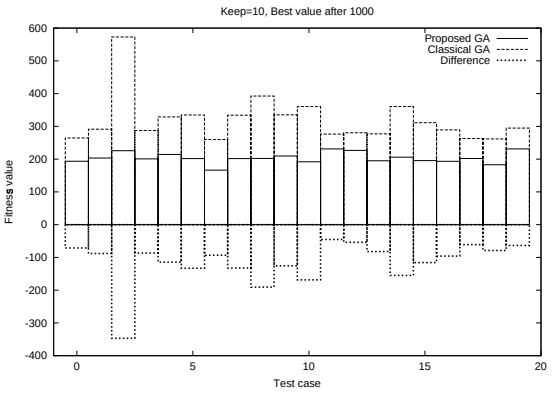


Figure 4: Best individuals fitness value after 1000 generation. (Small numerical value on fitness axis means a better solution.

4 Conclusion

The proposed method achieved a high improvement in the convergence speed without causing any genetic drift problem leading to a local minima. On the contrary the quality of the solution is usually better after a fixed number of iterations. Since the method converges faster, it is also much more suitable for time critical problems where a suboptimal solution is useful.

References

[1] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 1996.

[2] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[3] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

[4] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[5] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1975.

[6] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT press, 1996.

A Diversity-Control-Oriented Genetic Algorithm (DCGA) : Performance Improvement by the Reinitialization of the Population

Hisashi Shimodaira
Faculty of Information and Communications, Bunkyo University
2-2-16 Katsuradai, Aoba-Ku, Yokohama-City, Kanagawa 227-0034, Japan (E-mail: shimo-hi@hi-ho.ne.jp)

Abstract

In order to maintain the diversity of structures in the population and prevent premature convergence, I have developed a new genetic algorithm called DCGA. In the experiments on many standard benchmark problems, DCGA showed good performances, whereas with harder problems, in some cases, the phenomena were observed that the search was stagnated at a local optimum despite that the diversity of the population is maintained. In this paper, I propose methods for escaping such phenomena and improving the performance by reinitializing the population, that is, a method called each-structure-based reinitializing method with a deterministic structure diverging procedure as a method for producing new structures and an adaptive improvement probability bound as a search termination criterion. The results of experiments demonstrate that DCGA becomes robust in harder problems by employing these proposed methods and presents markedly superior performances to the previous leading GA in some problems.

1 INTRODUCTION

Genetic algorithms (GAs) are a promising means for function optimization. One problem plaguing traditional genetic algorithms is convergence to a local optimum. The genetic search process converges when the structures in the population are identical, or nearly so. Once this occurs, the crossover operator ceases to produce new structures, and the population stops evolving. Unfortunately, this often occurs before the true global optimum has been found. This behavior is called *premature convergence*. The cause of premature convergence is that the structures in the population are too alike. Therefore, one method for preventing premature convergence is to ensure that the different members of the population are different, that is, to maintain the diversity of structures in the population [1]. In order to achieve this goal, I have developed a new genetic algorithm called DCGA (Diversity-Control-oriented Genetic Algorithm) [2, 3, 4, 5, 6, 7].

In DCGA, the structures in the next generation are selected from the merged population of parents and their offspring with duplicates eliminated on the basis of a particular selection probability. The major feature of DCGA is that the distance between a structure and the best performance structure is used as the primary selection criterion and it is applied on the basis of a probabilistic function that produces a larger selection probability for a structure with a larger distance. The diversity of structures in the population can be externally controlled by adjusting the coefficients of the probability function so as to be in an appropriate condition according to the given problem.

Within the range of some experiments described in the previous papers [2, 3, 4, 5, 6], DCGA outperformed the simple GA and seems to be a promising competitor of the previously proposed algorithms such as Genitor [8] and CHC [9]. However, with harder problems, in some cases, the phenomena were observed that the search was stagnated at a local optimum despite that the diversity of the population is maintained. This paper proposes methods for escaping such phenomena and improving the performance by the reinitialization of the population.

The reinitialization (also often called restart) is that whenever the search cycle of the GA achieves its termination criteria, the structures in the population are reinitialized with or without the structures obtained so far to repeat the cycle. Relatively little work has been done in this area. Goldberg [10] proposed to generate a new population by transferring the best structures of the converged population to the new population and then generating the remaining structures randomly. Eshelman [9] employed in CHC a method of generating a new population by adding the best structure found so far to the new population and then generating the remaining structures by flipping a fixed portion of the bits of the best structure found so far that is chosen at random without replacement. Maresky [11] proposed a method of generating a new structure by reinitializing each bit of each structure with the probability of bit reinitialization. In evolutionary programming, Mathias [12] proposed a method of generating a new population by seeding one structure with the parameters of the best structure and then by initializing the parameter values for the remaining structures with values randomly chosen from a normal distribution centered around the corresponding parameter of the best structure. Fukunaga [13] proposed the use of a restart scheduling strategy which generates a static restart strategy with optimal expected utility, based on a database of past performance of the algorithm on a class of problem instances.

Common termination criteria used in practice include (1) cost bound: stop when a solution with lower than or equal to a given cost value is found, (2) time bound: stop after a given run time, (3) improvement probability bound (IPB): stop after no improvement had been found after some threshold number of generations, and (4) convergence bound: stop after the population seems to have converged [9, 11, 12].

In this paper, I propose a method called each-structure-based reinitializing method (ERM) with a deterministic structure

diverging procedure in which each structure in the new population is produced by flipping a fixed portion of the bits of each structure obtained so far. The flipping bits are chosen at random without replacement. Also, I propose an adaptive improvement probability bound (AIPB) in which the threshold number of generations is adaptively changed according to the diversity of the population measured by fitness values. Using harder benchmark problems, the performances of the following three kinds of reinitializing methods were tested: (1) new-structure-based reinitializing method (NRM), (2) best-structure-based reinitializing method (BRM) and (3) each-structure-based reinitializing method (ERM). In the former two methods, two cases where the best structure found so far is transferred or is not transferred to the new population are included. As the termination criterion, the IPB or the AIPB was used. The results showed that the performance of the ERM is superior to those of the other two methods and that AIPB is very effective in a function having local optima on the plateaus. As the results, the robustness of DCGA with the ERM and the IPB or the AIPB in harder problems was demonstrated.

2 OUTLINE OF DCGA

The skeleton of DCGA is shown in Fig. 1. The number of structures in the population P(t) is constant and denoted by N, where t is the generation number. The population is initialized by using uniform random numbers. In the selection for reproduction, all the structures in P(t-1) are paired by selecting randomly two structures without replacement to form P'(t-1). That is, P'(t-1) consists of N/2 pairs. By applying mutation with probability p_m and always applying crossover to the structures of each pair in P'(t-1), two offspring are produced and C(t) is formed. The mutation rate p_m is constant for all the structures. The structures in C(t) and P(t-1) are merged and sorted in order of their fitness values to form M(t). In the selection for survival, those structures that include the structure with the best fitness value are selected from M(t) and the population in the next generation P(t) is formed.

The details of the selection for survival, are as follows:

① Duplicate structures in M(t) are eliminated and M'(t) is formed. Duplicate structures mean that they have identical entire structures.

② Structures are selected by using the Cross-generational Probabilistic Survival Selection (CPSS) method, and P(t) is formed from the structure with the best fitness value in M'(t) and the selected structures. In the CPSS method, structures are selected by using uniform random numbers on the basis of a selection probability defined by the following equation:

$$p_s = \left\{ (1-c)\frac{h}{L} + c \right\}^{\alpha}, \qquad (1)$$

where h is the hamming distance between a candidate structure and the structure with the best fitness value, L is the length of the entire string representing the structure, c is the shape coefficient whose value is in the range of [0.0, 1.0], and α is the exponent. In the selection process, a uniform random number in the range of [0.0, 1.0] is generated for each structure. If the generated random number is smaller than p_s that is calculated by Eq.(1) for the structure, then the structure is selected; otherwise, it is deleted. The selection process is performed in order of the fitness values of all the structures in M'(t), without considering the fitness value of a structure itself, except the structure with the best fitness value.

③ If the number of the structures selected in the process ② is smaller than N, then new structures randomly generated as in the initial population are introduced by the difference of the numbers.

```
begin;
  t=0;
  initialize population P(t);
  evaluate structures in P(t);
  while (termination condition not satisfied) do;
  begin;
    t=t+1;
    select P'(t-1) from P(t-1) by randomly pairing all
      structures without replacement;
    apply mutation with p_m and crossover to each pair of
      P'(t-1) and produce two offspring to form C(t);
    evaluate structures in C(t);
    merge structures in C(t) and P(t-1) and sort them in
      order of their fitness values to form M(t);
    select N structures including the structure with the
      best fitness value from M(t) to form the next pop-
      ulation P(t) according to the following procedure:
      (1) eliminate duplicate structures in M(t) to form
          M'(t);
      (2) select structures from M'(t) with CPSS method
          in order of their fitness values;
      (3) if the number of selected structures is smaller
          than N, introduce new structures by the
          difference of the numbers;
  end;
end;
```

Fig. 1 The skeleton of DCGA

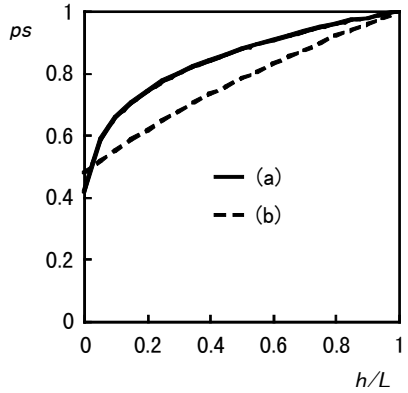


Fig. 2. Example curves of Eq. (1). (a) $\alpha = 0.19$, $c = 0.01$; (b) $\alpha = 0.5$, $c = 0.234$.

If the structure is represented as a bit string, the hamming distance between two structures can be calculated by the usual way. With a combinatorial optimization problem such as the traveling salesman problem, also, the extended hamming distance can be calculated as a minimum value of numbers of pairs that have different cities, when the cities of the two tours are paired each other in order of their path representations [2, 3, 4, 5]. DCGA is applicable to all sorts of optimization problems by using the definition of a proper distance measure between two structures.

The reasons for employing the above methods in DCGA are as follows.

As long as the structure with the best fitness value does not reach the global optimum, it is a local optimum. If the selective pressure for better-performing structures is too high, structures similar to the best-performing structure will increase in number and eventually take over the population. This situation is premature convergence, if it is a local optimum. Therefore, we need to reduce appropriately the selective pressure in the neighborhood of the best-performing structure to thin out structures similar to it. Eq. (1) can work to do such processing. Example curves of Eq. (1) are shown in Fig. 2. The selection of structures on the basis of Eq. (1) is biased toward thinning out structures with smaller hamming distance from the best-performing structure and selecting structures with larger hamming distances from the best-performing structure. As a result, the population is composed of various structures as demonstrated in Section 3. The larger bias produces the greater diversity of structures in the population. The degree of this bias is "externally" adjusted by the values of c and α in Eq. (1). Their appropriate values need to be explored by trial and error according to the given problem. As demonstrated in the experiments described later, Eq. (1) is very suitable for controlling the diversity of the structures in the population so as to be in an appropriate condition by adjusting the values of c and α.

In the selection for survival, the fitness values of the structures themselves are not considered. However, this does not mean to neglect the selective pressure. Because the selection process is performed in order of the fitness values of the structures and better-performing structures can have an appropriate chance to be selected, as a result, there exists an appropriate selective pressure determined by the value of the selection probability.

We can produce an appropriate selective pressure according to the given problem. That is, for a simple function with few local optima, higher selective pressure can be produced with a larger value of c and / or a smaller value of α For a complicated function with many local optima, lower selective pressure can be produced with a smaller value of c and / or a larger value of α. The selection with c = 1.0 in DCGA is the same as (N + N)-selection in the evolution strategies and the population-elitist selection in CHC [9].

In DCGA, structures that survived and the structure with the best fitness value obtained so far can always become parents and produce their offspring. Crossovers are always applied to diverse structures maintained in the population. When a pair of structures with a small distance are mated, their neighborhood can be examined to result in the local search. When a pair of structures with a large distance are mated, a region not yet explored can be examined to result in the global search. In such a way, local as well as global searches can be performed in parallel.

A shortcoming of DCGA is that the number of parameters to be tuned is three (mutation rate, and α and c in Eq. (1)) and they must be tuned trial and error according to the given problem. However, this is not a peculiar problem to DCGA, because Hart [14] has demonstrated theoretically that no single robust parameter set exists that is suitable for a wide rang of functions.

3 PERFORMANCE AND ISSUES OF DCGA IN HARDER PROBLEMS

The performance of DCGA for 13 standard benchmark functions were tested [6]. For 11 functions among them, the global optimums were obtained successfully in all runs, whereas for the Griewank and expanded Rosenbrock functions, the rates of successful runs were not 1.0. The expanded Rosenbrock function is as follows:

$$F_2 = \sum_{i=1}^{n-1} \left[100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2 \right] + 100(x_n^2 - x_1)^2 + (1 - x_n)^2 \qquad (2)$$

In this paper, therefore, I investigated the results for these two functions that are known as hard problems for GA to solve. The binary string representing a structure was transformed to real numbers in the phenotype so that for each coordinate, the corresponding real number matches exactly the coordinate value which gives the global optimum of the function. Exact global optimums in these functions were explored with the optimality threshold considering only round-off errors. The dimension of the problem (n) and the maximum number of function evaluation (*MXFE*) were set according to the previous studies [15]. Gray coding was used. Bit-flip mutation and uniform crossover HUX [9] were used. I performed 30 runs per parameter set, changing seed values for the random number generator to initialize the population. The run was continued until the global optimum was attained by at least one structure (I call this the success) or until *MXFE* was reached. The combination of best-performing parameter values including the population size was examined by changing their values little by little.

The performance was evaluated by the rate of successful runs out of the total runs (*SCR*) and the average value of function evaluation numbers in the successful runs (*AVFE*). Table 1 shows the definitions of major symbols used in the subsequent Tables. The best results of the experiments are summarized in Table 2.

In order to understand how DCGA succeeds or fails in attaining the global optimum during the search process, we need to examine how DCGA works and of what structures the population is composed. Thus, in some cases where DCGA succeeded or failed in attaining the global optimum, I examined the relationships between minimum (best), average, and maximum fitness values and generation number, and relationships between minimum, average, and maximum values of the ratio h / L and generation number.

Fig. 3 and Fig. 4, respectively, show the case where DCGA succeeded or failed in attaining the global optimum for the Griewank function (F_1, n = 10) under the condition shown in Table 2. In both cases, the diversity of the population in both genotypes and fitness values is maintained during the search. At the final stage, the best structures were trapped at local minimums. In the case of success, the solution could escape from the local minimum (0.0498) and reach the global minimum (0.0), whereas in the case of failure, the solution kept trapped at a local minimum (0.0488) until *MXFE*.

Table 2. Best results for the Griewank and expanded Rosenbrock functions without using reinitializing method.

F.	n	*MXFE*	N	α	c	p_m	*SCR*	*AVFE*	*SDFE*	*AVBF*
F_1	10	500000	46	0.21	0.01	0.006	0.87	160298	122713	$7.14 \cdot 10^{-3}$
	20	1000000	50	0.21	0.0096	0.0021	0.67	306324	202106	$1.93 \cdot 10^{-2}$
F_2	6	500000	28	0.2	0.008	0.012	0.53	77723	24167	2.77
	8	1000000	34	0.204	0.0005	0.01	0.5	238829	116860	3.96
	10	5000000	42	0.2	0.002	0.011	0.47	2286790	812327	5.28

Table 1. Definitions of symbols in the subsequent Tables.

Symbol	Definition
F_1	Griewank function.
F_2	Expanded Rosenbrock function.
N	Population size.
n	Number of dimension.
α	Exponent for probability function, Eq. (1).
c	Shape coefficient for probability function, Eq. (1).
STC	Search termination criterion. IPB: Improvement probability bound, AIPB: Adaptive improvement probability bound.
PRM	Population reinitializing method
SDP	Structure diverging procedure. D: deterministic, P: probabilistic.
g_0	Fixed threshold number of generations for reinitializing population in Eq. (3).
k	Coefficient to adjust increasing rate in Eq. (3).
DVR	Divergence rate for generating new individuals.
SCR	Success rate (rate of successful runs).
AVFE	Average value of function evaluation numbers in the successful runs.
SDFE	Standard deviation of function evaluation numbers in the successful runs.
AVBF	Average value of best fitness values in all runs.
MXFE	Maximum value of function evaluation numbers.

Fig. 5 and Fig. 6, respectively, show the case where DCGA succeeded or failed in attaining the global optimum for the expanded Rosenbrock function (F_2, n = 6) under the condition shown in Table 2. In the case of success, the diversity of the population in both genotypes and fitness values is maintained during the search. Although at the final stage, the best structures were trapped at a local minimum, the solution could escape from the local minimum ($5.01 \cdot 10^{-4}$) and reach the global minimum (0.0). In the case of failure, the diversity of the population in genotypes is maintained during the search, whereas that in fitness values is lost at the early stage. This means that all the structures of the population lie on a plateau and the best structure could not escape from a local minimum (5.94) on it.

The cases of failure in both functions indicate that it is vain attempting to escape from some kind of local minimum by extensive search, because crossover and bit-flip mutation operators can not produce a structure that can escape from it despite that the diversity of the population in genotype is maintained.

4 METHODS FOR REINITIALIZING POPULATION

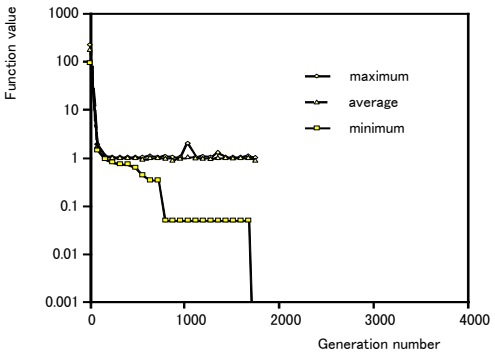
The reinitialization (also often called restart) is that whenever the search cycle achieves its termination criteria, the population are reinitialized with or without the structures obtained so far to repeat the cycle. This is based on the experience that rather than attempting to escape from a local optimum by extensive search, it is better to terminate the search and restart from a new initial state in order to attain the global optimum within limited computation time.

In this paper, I examined the following three methods for reinitializing the old population and producing a new population.

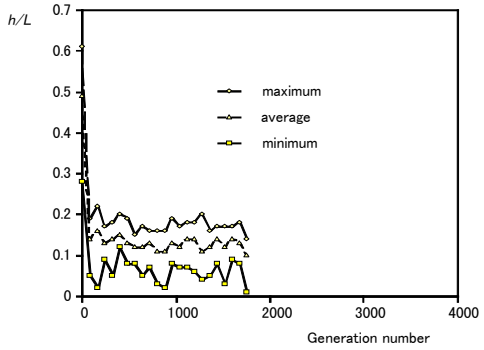
(1) New-structure-based reinitializing method: In this method, a new population is produced by generating all the structures randomly (NRM), or, a new population is produced by transferring the best structure found so far to the new population (elitist strategy) and then generating the remaining structures randomly (NRM_E) as suggested by Goldberg [10].

(2) Best-structure-based reinitializing method: The basic strategy of this method is using the best structure found so far as a template to generate a structure in the new population. In the deterministic structure diverging procedure, each structure is produced by flipping a fixed portion of the bits of the best structure as in CHC [9]. The fixed portion is chosen at random without replacement. The rate of the fixed portion to the length of the string is called divergence rate (*DVR*). In the probabilistic structure diverging procedure, each bit of each structure is produced by transferring the bit of the best structure or generating it randomly with the divergence rate. That is, when the value of a random number calculated for the bit is smaller than the value of the divergence rate, the value of the bit is generated by using uniform random number; otherwise, the bit of the best structure is transferred to that of the structure. A new population is produced by generating all the structures by these procedure (BRM), or, a new population is produced by transferring the best structure to the new population and then generating the remaining structures by these procedure (BRM_E).

(3) Each-structure-based reinitializing method (ERM): This method is based on the hypothesis that it is better to use all the structures in the old population in order to explore regions not yet explored, because all the structures contain some information on the search space. Thus, the basic strategy of this method is using each structure in the old population as a template to generate each structure in the new population. In the deterministic structure diverging procedure that I propose in this paper, each structure is produced by flipping a fixed portion of each structure in the old population in the same way as described in (2). The fixed portion is chosen at random without replacement. In the probabilistic structure diverging procedure proposed by Maresky [11], each bit of each structure is produced by transferring the bit of each

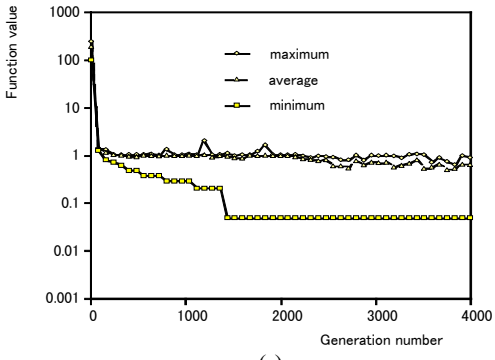


(a)

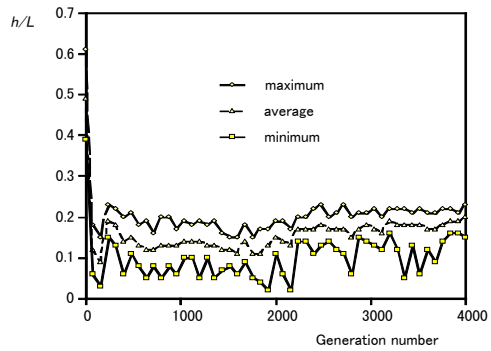


(b)

Fig. 3 (a) Function value vs. generation number and (b) h/L vs. generation number in a successful run for Griewank function (F_1).

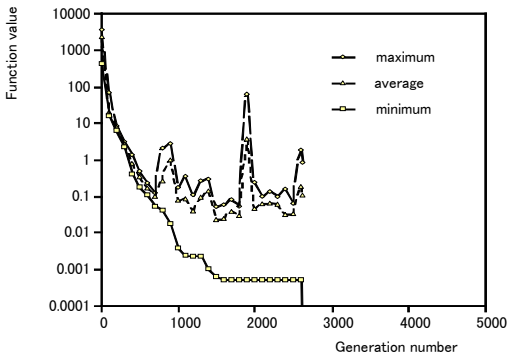


(a)

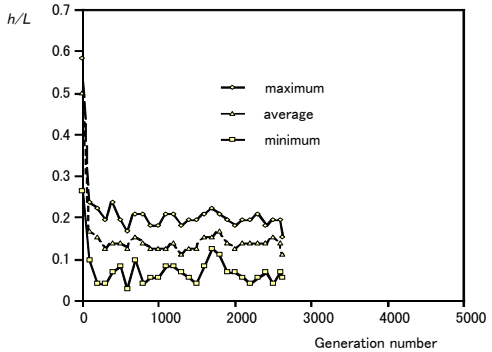


(b)

Fig. 4 (a) Function value vs. generation number and (b) h/L vs. generation number in a failure run for Griewank function (F_1).

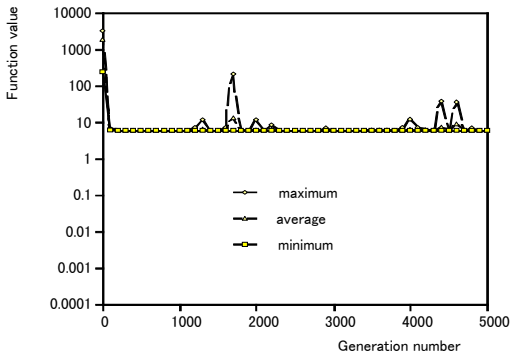


(a)

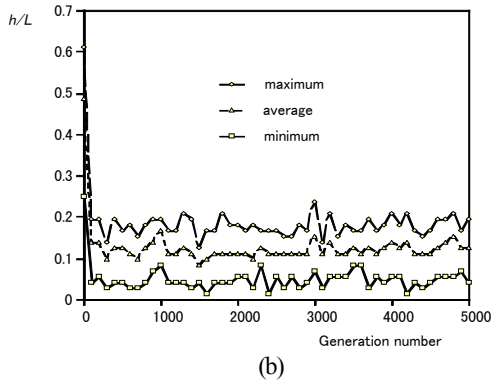


(b)

Fig. 5 (a) Function value vs. generation number and (b) h/L vs. generation number in a successful run for Rosenbrock function (F_2).



(a)



(b)

Fig. 6 (a) Function value vs. generation number and (b) h/L vs. generation number in a failure run for Rosenbrock function (F_2).

structure or generating it randomly with the divergence rate in the same way as described in (2). The elitist strategy is not adopted in this method.

Table 3 shows the acronyms of the population reinitializing methods.

The optimum point to terminate search and restart is determined empirically over a range of problem size and complexities. The most often used method that is called improvement probability bound (IPB) [13] is to terminate the search if after a certain number of generations no better solutions have been found (that is, the best fitness value has not changed). I propose an adaptive improvement probability bound (AIPB). In this method, the threshold number of generations is adaptively changed according to the diversity of the population measured by fitness values on the basis of the following equation:

$$g = g_0 \left(1 + k \frac{f_{\max} - f_{\min}}{| f |} \right), \qquad (3)$$

Table 3. Acronyms of population reinitializing methods.

Acronym	Method
NRM	New-structure-based reinitializing method without elitist strategy
NRM_E	New-structure-based reinitializing method with elitist strategy
BRM	Best-structure-based reinitializing method without elitist strategy
BRM_E	Best-structure-based reinitializing method with elitist strategy
ERM	Each-structure-based reinitializing method without elitist strategy

where g is the adaptively changed threshold number of generations; g_0 is the fixed threshold number of generations; k is the coefficient to adjust the increasing rate; f_{max} is the maximum fitness value of the structures; f_{min} is the minimum fitness value of the structures; and $| f |$ is the larger value of $|f_{max}|$ and $|f_{min}|$. In this equation, $(f_{max} - f_{min})$ represents the diversity of the population measured by fitness values and is divided by $| f |$ in order to normalizing it. With this equation, the threshold number of generations is adaptively adjusted within the range between g_0 and $g_0 (1 + k)$ according to the diversity of the population measured by fitness values. The search is continued longer, when the diversity of the population measured by fitness values is larger.

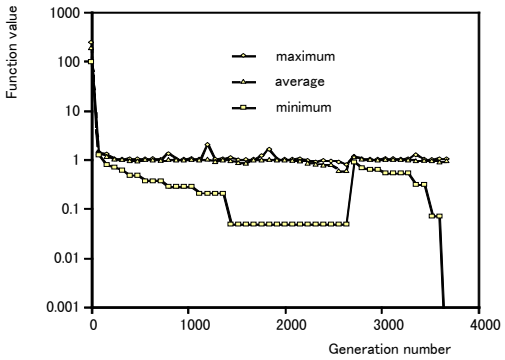
5 EXPERIMENTS AND THE RESULTS

The performances of the population initializing methods above mentioned were tested using the Griewank function and extended Rosenbrock functions. The computation condition including all the values of the control parameters (N, α, c, p_m) are the same as in the experiments described in Section 3. The combination of best-performing values of g_0, k and DVR were examined by changing their values little by little.

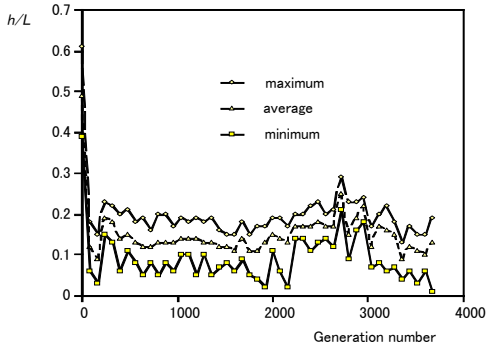
Table 4 shows the best results for the Griewank function. For the case n = 10 and STC = IPB, the performances of all the population initializing methods were examined. According to the results, the performances of the deterministic and probabilistic structure diverging procedures are almost the same. Among those methods, the ERM with deterministic structure diverging procedure shows the best performance that SCR = 1.0 and the performance is improved slightly by using AIPB as STC. With the case n = 20 and STC = IPB, the ERM with deterministic structure diverging procedure also shows the good performance

Table 4. Best results for the Griewank (F_1) and expanded Rosenbrock (F_2) functions using each reinitializing method

F.	n	STC	PRM	SDP	g_0	k	DVR	SCR	$AVFE$	$SDFE$	$AVBF$
F_1	10	IPB	NRM	—	1500	—	—	0.97	179697	121161	$3.29 \cdot 10^{-3}$
			NRM_E	—	2000	—	—	0.83	124710	62407	$8.18 \cdot 10^{-3}$
			BRM	D	1000	—	0.45	0.97	183175	122181	$9.16 \cdot 10^{-3}$
			BRM_E	D	2000	—	0.6	0.83	132859	78971	$8.74 \cdot 10^{-3}$
			ERM	D	1250	—	0.33	1.00	157312	94668	0.0
			BRM	P	1000	—	0.55	0.97	173253	120117	$1.66 \cdot 10^{-3}$
			BRM_E	P	1000	—	0.55	0.8	141399	93730	$1.00 \cdot 10^{-2}$
			ERM	P	1000	—	0.25	1.00	160743	101743	0.0
		AIPB	ERM	D	740	0.7	0.33	1.00	147553	87345	0.0
	20	IPB	ERM	D	1800	—	0.3	1.00	419769	236028	0.0
F_2	6	IPB	NRM	—	1200	—	—	0.93	171927	109847	$9.70 \cdot 10^{-4}$
			NRM_E	—	1100	—	—	0.97	177490	125987	$1.98 \cdot 10^{-1}$
			BRM	D	1400	—	0.74	1.00	120519	77350	0.0
			BRM_E	D	1400	—	0.80	1.00	149064	99641	0.0
			ERM	D	1400	—	0.81	1.00	118365	72187	0.0
			BRM	P	1200	—	1.00	0.93	166721	118755	$1.98 \cdot 10^{-1}$
			BRM_E	P	1100	—	1.00	0.87	152495	107923	$3.96 \cdot 10^{-1}$
			ERM	P	1200	—	1.00	0.93	161627	117855	$1.98 \cdot 10^{-1}$
		AIPB	ERM	D	500	2.0	0.80	1.00	94850	40070	0.0
	8	IPB	ERM	D	4000	—	0.80	1.00	443861	261869	0.0
		AIPB	ERM	D	1900	2.4	0.80	1.00	323627	143343	0.0
	10	IPB	ERM	D	35000	—	0.80	0.83	3129242	1257473	$6.60 \cdot 10^{-1}$
		AIPB	ERM	D	10000	4.0	0.85	1.00	2547265	743240	0.0

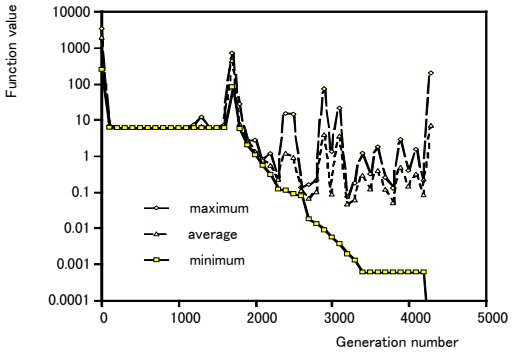


(a)

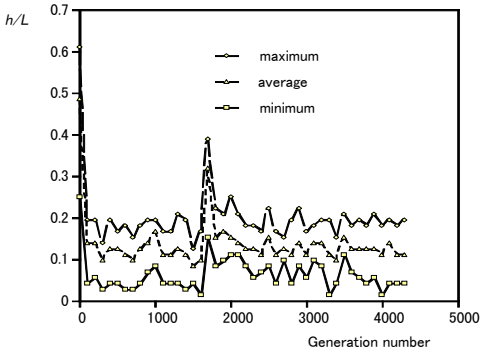


(b)

Fig. 7. (a) Function value vs. generation number and (b) h / L vs. generation number for the Griewank function (F_1) with ERM and IPB.

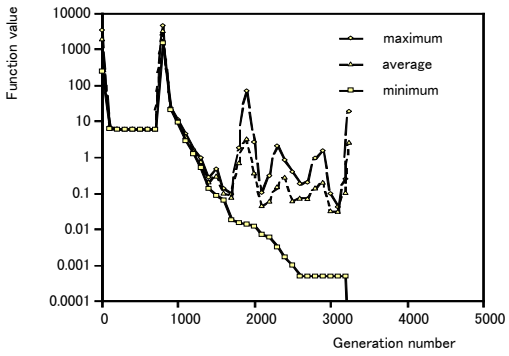


(a)

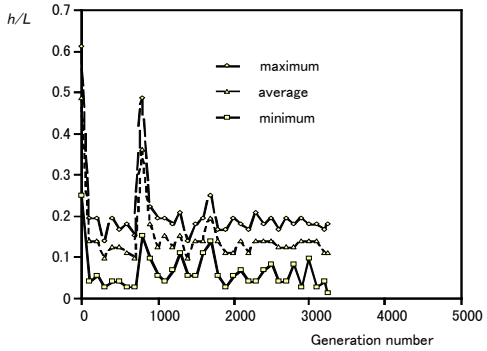


(b)

Fig. 8. (a) Function value vs. generation number and (b) h / L vs. generation number for the Rosenbrock function (F_2) with ERM and IPB.



(a)



(b)

Fig. 9. (a) Function value vs. generation number and (b) h / L vs. generation number for the Rosenbrock function (F_2) with ERM and AIPB.

that $SCR = 1.0$.

Table 4 also shows the best results for the expanded Rosenbrock function. For the case $n = 6$ and STC =IPB, the performances of all the population initializing methods were examined. According to the results, the performances with the deterministic structure diverging procedure are markedly better than those with the probabilistic structure diverging procedure. Among those methods, the ERM with deterministic structure diverging procedure shows the best performance that $SCR = 1.0$ and the performance is improved greatly by using the AIPB as the STC. With the cases $n = 8$ and $n = 10$ using the ERM with the deterministic structure diverging procedure, the performances with the AIPB shows the good performance that $SCR = 1.0$ and are markedly superior to those with the IPB.

In order to demonstrate the effect of the population reinitializing method, I examined the change of diversity of the population in genotypes and fitness values during the search, for the case where DCGA without the population reinitializing method failed in attaining the global optimum, whereas DCGA with the population reinitializing method succeeded in attaining the global optimum using the same initial condition. For the Griewank function with PRM = ERM, STC = IPB and SDP = D, Fig. 7 demonstrates the result in the same case as shown in Fig. 4, where the solution can escape from the local minimum (0.0488) by initializing the population to reach the global optimum. For the expanded Rosenbrock function with PRM = ERM, STC = IPB and SDP = D, Fig. 8 demonstrates the result in the same case as shown in Fig. 6, where the solution can escape from the local minimum (5.94) on the plateau by initializing the population to reach the global optimum. Fig. 9 shows that in this case, by using the AIPB instead

of the IPB as the search termination criterion, the period of the search of the local optimum on the plateau is shortened to result in improving the performance.

6 DISCUSSIONS

The results of the above experiments are summarized as follows:

(1) As for the SDP, as a rule, the deterministic structure diverging procedure is superior to the probabilistic structure diverging procedure.

(2) As a rule, the performances with elitist strategy are inferior to those without elitist strategy. This is partly because the best structure is apt to take over the population again, before the remaining structures evolve to still better structures.

(3) With the deterministic structure diverging procedure, the performances of the ERM are better than those of the NRM. Additionally, the performances of the ERM are better than those of the BRM. For both the functions, the ERM with deterministic structure diverging procedure proposed in this paper shows the best performance. Also, the optimum values of the divergence rate are relatively large. These justify the hypothesis for this method that it is better to use all the structures in the old population in order to explore regions not yet explored, because all the structures contain some information on the search space. The ERM can produce the initial search points to escape from the local optimum at which the population was trapped with relatively large divergence rates.

(4) In the expanded Rosenbrock function, the performance is improved greatly by using the AIPB proposed in this paper instead of the IPB as the STC. This demonstrates that the AIPB is effective for a function with local minimums on plateaus.

(5) For both the functions with different dimensions, the ERM with deterministic structure diverging procedure using the IPB or the AIPB shows the performance that *SCR* = 1.0. The performances of DCGA with these methods are markedly superior to those of DCGA without the population reinitializing method.

In conclusion, DCGA becomes robust in harder problems by employing these proposed methods and presents markedly superior performances to the previous leading GA in some problems. For example, this is true of CHC for the expanded Rosenbrock function with *MXFE* = 5000000, because its success rates are 0.0, 0.067 and 0.033 for n = 6, 8, 10, respectively [15].

7 CONCLUSIONS

Within the range of the above experiments, the following conclusions can be drawn.

(1) The ERM with deterministic structure diverging procedure proposed in this paper shows the best performance and the optimum values of the divergence rate are relatively large. Therefore, The ERM can produce the initial search points to escape from the local optimum at which the population was trapped with relatively large divergence rates.

(2) The AIPB proposed in this paper is effective for a function with local minimums on plateaus and the performance is improved greatly by using the AIPB instead of the IPB as the search termination criterion.

(3) The performances of DCGA with the proposed methods are markedly superior to those of DCGA without the population reinitializing method.

(4) DCGA becomes robust in harder problems by employing these proposed methods and presents markedly superior performances to the previous leading GA in some problems.

REFERENCES

[1] M. L. Mauldin, "Maintaining Diversity in Genetic Search," in *Proc. of the National Conference on Artificial Intelligence*, 1984, pp. 247-250.

[2] H. Shimodaira, "DCGA: A Diversity Control Oriented Genetic Algorithm," in *Proc. of the Second IEE/IEEE Int. Conference on Genetic Algorithms in Engineering Systems (GALESIA'97)*, 1997, pp. 444-449.

[3] H. Shimodaira, "DCGA: A Diversity Control Oriented Genetic Algorithm," in *Proc. of the Ninth IEEE Int. Conference on Tools with Artificial Intelligence*, 1997, pp. 367-374.

[4] H. Shimodaira, "A Diversity-Control-Oriented Genetic Algorithm (DCGA): Development and Initial Results," Transactions of Information Processing Society of Japan, Vol. 40, No. 6, 1999.

[5] H. Shimodaira, "A Diversity-Control-Oriented Genetic Algorithm (DCGA): Development and Experimental Results," in *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-99) Volume 1*, Morgan Kaufmann, 1999, pp.603-611.

[6] H. Shimodaira, "A Diversity-Control-Oriented Genetic Algorithm (DCGA): Performance in Function Optimization," in *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Morgan Kaufmann, 2000, p. 366.

[7] http://www.hi-ho.ne.jp/shimo-hi/new_ga.htm

[8] D. Whitley, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproduction Trials is Best", in *Proc. of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989, pp. 116-121.

[9] L. J. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," in *Foundation of Genetic Algorithms*, G. J. E. Rawlins Ed., California: Morgan Kaufmann, 1991, pp. 265-283.

[10] D. E. Goldberg, "Sizing Populations for Serial and Parallel Genetic Algorithms", in *Proc. of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989, pp. 70-79.

[11] J. Maresky, Y. Davidor and D. Gitler, "Selectively Destructive Re-start", in *Proc. of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1995, pp. 144-150.

[12] K. E. Mathias, J. D. Schaffer and L. J. Eshelman, "The Effects of Control Parameters and Restarts on Search Stagnation in Evolutionary Programming", in *Parallel Problem Solving from Nature - PPSN V*, Springer, 1998, pp. 398-407.

[13] A. S. Fukunaga, "Restart Scheduling for Genetic Algorithms", in *Parallel Problem Solving from Nature - PPSN V*, Springer, 1998, pp.357-366.

[14] W. E. Hart and R. K. Belew, "Optimizing an Arbitrary Function is Hard for the Genetic Algorithm", in *Proc. of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991, pp. 190-195.

[15] D. Whitley, K. Mathias, S. Rana and J. Dzubera, "Building Better Test Functions," in *Proc. of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1995, pp. 239-246.

On Biologically Inspired Genetic Operators: Transformation in the Standard Genetic Algorithm

Anabela Simões[1,2]
[1]Instituto Superior de Engenharia de Coimbra
Quinta da Nora
3030 Coimbra - Portugal
abs@isec.pt

Ernesto Costa[2]
[2]Centro de Informática e Sistemas da Universidade de Coimbra, Polo II - Pinhal de Marrocos
3030 Coimbra - Portugal
ernesto@dei.uc.pt

Abstract

In this paper, we introduce a biologically inspired recombination operator that occurs in the colonies of bacteria. The mechanism is called transformation and is responsible for the genetic variation and consequently the advantageous characteristics that some bacteria possess. We present an implementation of the transformation mechanism in the standard GA (SGA) and we compare its performance solving two different classes of problems using either transformation or the traditional crossover operators. The results show that the GA using transformation is always superior to the SGA. The good results obtained by transformation seem to be related to the great degree of diversity that the mechanism introduces in population.

1 INTRODUCTION

For a population to survive changes in its environment it must have sufficient genetic variety to adapt to the new conditions: less genetically diverse populations may be at greater risk. Known as genetic diversity, this great variation within species is what allows populations to adapt to changes in climate and other local environmental conditions.

Genetic Algorithms (GAs) are inspired by genetics and natural selection: a population evolves through a number of generations, where the fittest individuals are more likely to be selected to reproduce in each generation. This process allows the evolution of the population to the best solution (Holland, 1992; Goldberg, 1989). The population's diversity is introduced by the application of two main genetic operators: mutation and crossover. These operators produce changes in the individuals, creating evolutionary advantages in some of them.

Nature maintains genetic diversity by several mechanisms besides crossover and mutation. Some of those mechanisms are: inversion, transduction, transformation, conjugation, transposition and translocation (Gould and Keeton, 1996).

Some researchers in the field of Evolutionary Computation (EC) highlighted the importance of studying different biologically inspired genetic operators. (Mitchell and Forrest, 1994) and (Banzhaf et al., 1998) stress that it would be important to analyze if some of the mechanisms of rearranging genetic material present in the biological systems, when implemented and used in the Evolutionary Algorithms (EA), improve their performance.

Several authors have already used some biologically inspired mechanisms besides crossover and mutation in EA. For instance, inversion (Holland, 1992), conjugation (Harvey, 1996), translocation (De Falco et al., 2000), transduction (Nawa et al., 1999) and transposition (Simões and Costa, 1999; 2001a) were already used as the main genetic operators in the EA. As far as we know none implementation of the transformation mechanism was tested in EA.

Bacteria sometimes take up and incorporate fragments of DNA from the environment. This is called transformation (Clark and Russell, 1997).

In this paper, we propose a computational implementation of the transformation mechanism and we study the GA performance solving two different problems. The empirical analysis will focus the application of the traditional crossover operators and transformation, for different population's size. The two classes of problems used to study the GA performance were function optimization (Rastrigin, Griewangk, Schwefel and Ackley test functions) and a combinatorial optimization problem (0/1 knapsack problem).

This paper is organized in the following manner. First, in section 2, we describe the biological functioning of the transformation mechanism and we introduce our computational implementation for the proposed recombination mechanism. Section 3, details the characteristics of the experimental environment, including the selected problems to test the GA performance and the GA parameters. In section 4, we make an exhaustive

comparison of the results obtained with the proposed recombination operator and with the standard crossover operators (1-point, 2-point and uniform crossover). Finally, we present the relevant conclusions of the work.

2 TRANSFORMATION

In our work, we will propose a modified GA with the introduction of a new biologically inspired operator, called transformation. Next sections will describe this mechanism.

2.1 BIOLOGICAL TRANSFORMATION

Some bacteria readily take up outside DNA. If they have this ability, they are said to be competent. Competent bacteria can absorb fragments of DNA proceeding from dead bacteria and present in their environment.

Usually, transformation consists in the transfer of small pieces of extra cellular DNA between organisms. These strains of DNA, or gene segments, are extracted from the environment and added to recipient cells (Russell, 1998).

After that, there are two possibilities, failure or success, known technically as restriction and recombination. Restriction is the destruction of the incoming foreign DNA, since those bacteria assume that foreign DNA is more likely to come from an enemy, such as a virus. In this case, transformation fails. Recombination is the physical incorporation of some of the incoming DNA into the bacterial chromosome. If this happens, genes from the assimilated segment replace some of the host cell's genetic information and bacteria are permanently transformed. Once integrated in the chromosome, the DNA segment is able to survive.

2.2 COMPUTATIONAL TRANSFORMATION

The DNA fragments to incorporate in the individuals of the population are generated at the beginning of the process. This DNA fragments consist in binary strings of different lengths and will form the gene segment pool.

We will use the transformation mechanism as the main genetic operator in the GA. Therefore, transformation is applied every generation instead of the standard crossover operator. First, we select the individuals to be transformed using the roulette-wheel selection method and these individuals are changed with a fixed probability. Part of the gene segment pool is changed every generation, using genetic information of the individuals of the population.

This modified GA will be referred as Transformation-based GA (TGA) and is described in Figure 1.

The main aspects to consider in the implementation of transformation are the origin of the gene segments that will transform each individual and how the process of transformation will occur. These aspects will be detailed in the next sections.

1. Generate Initial Population
   Generate Initial Gene Segment Pool
2. DO
   2.1. Evaluate Population
   2.2. Select Individuals
   2.3. Transform Individuals
   2.4. Replace Population with New Individuals
   2.5. Create New Gene Segment Pool
WHILE (NOT Stop_Condition)

Figure 1: The GA Modified with Transformation

2.2.1 The Basic Functioning of the Transformation Mechanism

The GA starts with an initial population of individuals and an initial pool of gene segments, both created at random. In each generation, we select individuals to be transformed and we modify them using the gene segments in the segment pool. After that, the segment pool is changed, using the old population to create part of the new segments with the remaining being created at random (see Figure 2).

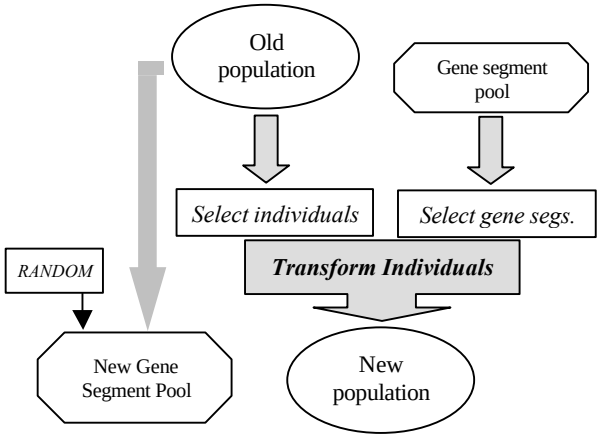


Figure 2: Computational Transformation

2.2.2 Origin of the Gene Segments

The segments that each individual will take up from the "surrounding environment" will proceed, mostly, from the individuals existing in the previous generation. In the used experimental setup, we changed the segment pool every generation. The modifications were made replacing 70% of the segments with new ones, created from the individuals of the old population. The remaining 30% were created at random. The size of the gene segments is also chosen in a random manner.

2.2.3 Transforming the Genetic Information of an Individual

After selecting individuals to a mating pool, we use the transformation mechanism to produce new individuals. In this case, there is no sexual reproduction among the individuals of the population. Each individual will generate a new one through the process of transformation. We can consider this process a form of asexual reproduction. Each individual will be transformed using a transformation probability.

The proposed mechanism can be described as follows: we select a segment from the segment pool and we randomly choose a point of transformation in the selected individual. The segment is incorporated in the genome of the individual, replacing the genes after the transformation point, previously selected. Obviously, the chromosome is seen as a circle. Proceeding this way the chromosome length is kept constant. This corresponds to the biological process where the gene segments, when integrated in the recipient's cell DNA, replace some genes in its chromosome. Figure 3 illustrates the process of transforming an individual.

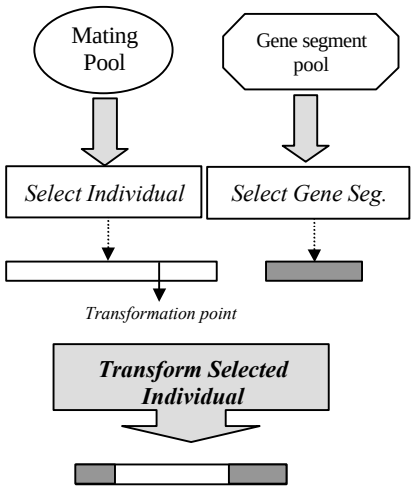


Figure 3: Transforming an Individual

3 EXPERIMENTAL SETTINGS

In order to investigate the performance of the TGA, we selected two different classes of problems: a combinatorial optimization problem (the 0/1 Knapsack problem (KP)) and the function optimization domain.

We selected these problems, since that, they are well known benchmarks to EA (Goldberg, 1989).

3.1 THE 0/1 KNAPSACK PROBLEM

The knapsack problem is a NP-complete problem, where we have to find the feasible combination of objects so that the total value of the objects put in the knapsack is maximized, subject to a capacity or weight constraint.

Formally, let C be the weight limitation (maximum permissible weight of the knapsack), let the integers 1,2,…. n denote n available types of objects, p_i and w_i, the value (or profit) and the weight of the ith object, respectively. A solution for the problem is represented by the binary vector **x** of length n. Each element of **x** can be zero or one: if x_i=1 then the item i was selected for the knapsack.

The knapsack problem can be expressed as

$$max \sum_{i=1}^{n} p_i.x_i \quad (1)$$

i.e., maximizing the profits, subject to the weight constraint

$$\sum_{i=1}^{n} x_i.w_i \le C, \quad (2)$$

where x_i is the selected object.

3.1.1 The Implemented Knapsack

We used several knapsack types (with 50, 100, 250 and 500 items). The evaluation of the solutions used a penalty function; the weights and profits vectors were created without any correlation and we used average capacity for the knapsack, as suggested in (Michalewicz, 1999).

The fitness f(x) for each binary string is determined as:

$$f(x) = \sum_{i=1}^{n} x_i.p_i - Pen(x) \quad (3)$$

with $Pen(x)$ the penalty function.

The penalty function is zero to all feasible solutions (those that don't exceed the knapsack capacity) and greater than zero otherwise. There are many possibilities for assigning the penalty value to the infeasible solutions. In our case, we considered a logarithmic penalty function defined by expression (4).

$$Pen(x) = log_2\left(1 + r.\left(\sum_{i=1}^{n} x_i.w_i - C\right)\right) \quad (4)$$

with $r = max_{i=1..n}\{p_i/w_i\}$

The generation of the vectors of profits (P[i]) and weights (W[i]) was made using the uncorrelated method, i.e.,

W[i]=(uniformly) random ([1..v])

P[i]=(uniformly) random ([1..v])

The value used for the parameter v was 10.

The capacity of the knapsack (average capacity) was calculated by:

$$C = 0.5\sum_{i=1}^{n} W[i] \quad (5)$$

3.2 TEST FUNCTIONS

We also evaluate the transformation mechanism by comparing its performance with the performance of the SGA (using three standard crossover operators) on several function optimization problems. To assess the quality of the algorithms we used the minimum function value found after a fixed number of function evaluations (50000, 100000 and 200000 in this case). The selected functions selected to analyze the GA performance were *Rastrigin*, *Schwefel, Griewangk*, and *Ackley* functions. All those functions are highly multimodal and have been used in other experimental comparisons of EA (Potter and De Jong, 1994; Gordon and Whitley, 1993).

The Rastrigin function is defined as:

$$f(x) = A*n + \sum_{i=1}^{n} x_i^{\,2} - A*\cos(2\Pi x_i) \qquad (6)$$

where n=20, A=10 and $-5.12 \le x_i \le 5.12$. The main characteristic of this function is the existence of many sub-optimal peaks whose values increase as the distance of the global optimum point increases

The Schwefel function is defined as:

$$f(x) = V*n + \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right) \qquad (7)$$

where n=10, V=418.9829 and $-500 \le x_i \le 500$. The global minimum of the function is zero. The interesting aspect of this function is the existence of a second-best minimum far away from the global minimum, which can trap the optimization algorithms on a local optimum.

The Griewangk function is defined as:

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^{\,2}}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \qquad (8)$$

where n=10 and $-600 \le x_i \le 600$. This function has a product term, which introduces interdependency among the variables.

The Ackley function is defined as:

$$f(x) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^{\,2}}\right) - \exp\left(\frac{1}{n}\cos(2\pi x_i)\right) \qquad (9)$$

where n=30 and $-30 \le x_i \le 30$. At a low dimension the landscape of this function is unimodal, however, the second exponential term covers the landscape with many small peaks and valleys.

3.3 THE PARAMETERS OF THE GENETIC ALGORITHM

The GA was first implemented with crossover (1-point. 2-point and uniform) and then with transformation. In the first problem, the 0/1 knapsack, we executed experiments to study the effect of the population size in the GA efficiency. Therefore, the population size varied between 20, 50, 100 and 200 individuals. In this problem, the GA evolved through 1000 generations.

For the function optimization domain, we fixed the maximum number of function evaluations equal to 200000.

In both classes of problems, we used binary representation to encode the problem, the roulette wheel selection and an elite size of two individuals. The mutation and crossover/transformation rate were 0.1% and 70%, respectively. The results reported in the next sections are the average computed over twenty-five runs.

3.4 EVALUATION MEASURE

We used the De Jong's off-line measure to compare GA efficiency when applied crossover or transformation (De Jong 1975). This measure is defined by:

$$X^*_e(g) = \frac{1}{T} * \sum_{t=1}^{T} f_e^{\,*}(t) \qquad (10)$$

where $f_e^{\,*} = best\ \{f_e(1), f_e(2), ..., f_e(n)\}$ and **T** is the number of runs. This means that off-line measure is the average of the best individuals in each generation. Due to the 25 trials, the average of the 25 runs was evaluated.

4 EXPERIMENTAL RESULTS

Next sections show the averaged results obtained in the knapsack problem and in the selected test functions.

4.1 RESULTS OBTAINED IN THE KNAPSACK PROBLEM

The proposed mechanism allowed the GA to achieve better solutions than the SGA using one-point, two-point or uniform crossover. This observation can be generalized to all the tested instances of the KP, i.e., with 50, 100, 250 and 500 items. Table 1 summarizes all the results for the 0/1 KP using the SGA and the TGA with different population's sizes. The best solutions found for n=50, 100, 250 and 500 are marked in bold.

As we can see, the population size is an important parameter when using crossover. In fact, increasing the population size from 50 to 100 or 200 individuals, crossover's performance shows some improvements. Using transformation with smaller populations, the GA obtained better results than the SGA with larger populations. As we can see in the table, with only 20 individuals in the population the TGA achieves solutions superior to the ones achieved by the SGA with 200 individuals.

Table 1: Summary of the Obtained Results using the SGA and the TGA with Different Population's Size

		Genetic Operator												
		One-point Crossover			*Two-point Crossover*			*Uniform Crossover*			*Transformation*			
	P. Size®	*50*	*100*	*200*	*50*	*100*	*200*	*50*	*100*	*200*	*20*	*50*	*100*	*200*
Nº of Items	50	438.63	482.18	465.20	461.41	494.41	496.37	488.74	533.54	507.24	562.99	568.62	574.76	**590.40**
	100	358.71	454.66	466.24	439.65	491.63	514.46	490.97	511.70	520.96	528.21	551.28	576.93	**590.40**
	250	950.08	1074.39	1089.54	923.92	1120.51	1036.18	1037.94	1211.10	1173.67	1330.68	1361.94	1375.79	**1410.28**
	500	1734.66	1985.96	1959.66	1845.88	1972.49	1996.18	2001.11	2303.35	2183.51	2548.29	2630.82	2633.53	**2656.17**

In order to understand these results, it is important to see how the GA evolved through the 1000 generations. In Figure 4, we show a representative example for the KP with 100 items. The figure compares the GA performance using uniform crossover with 200 individuals and transformation with a population of 50 binary strings.

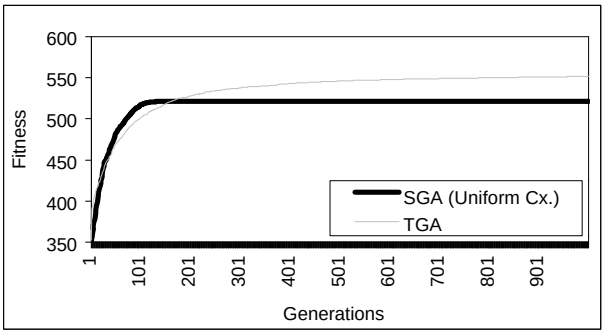


Figure 4: Comparing the SGA (200 inds.) and the TGA (50 inds.) Performances

As Figure 4 shows, uniform crossover only allow the SGA to improve in the first generations and after that the evolution stops. The TGA evolved during a long period, and was able to reach better results than crossover, even with a smaller population.

To analyze the influence of the population size in the GA's performance when using transformation we show, in Figure 5, the results obtained for the KP with 100 items. To the other instances, the results are quite similar. We can see that when using larger populations the maximum result obtained is superior.

Comparing the execution times spent by the four genetic operators solving the KP, we can see that transformation is the mechanism that consumes more time. Nevertheless, the differences are relatively small compared with the crossover operators. The time spent by the TGA is approximately 7% superior to the time spent by the operator that obtains the worst results (one-point crossover) and 3% superior to the best crossover operator (uniform crossover).

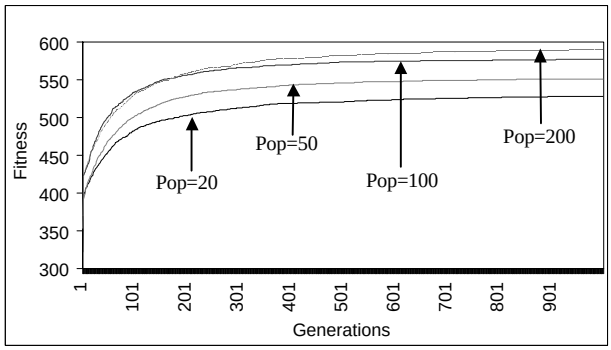


Figure 5: The TGA's Performance using Different Population's Size

Table 2 reports the results (in seconds) obtained running 25 trials of the SGA and the TGA with a population of 200 individuals, in a Pentium II with a 300 MHz processor.

Table 2: Time Spent to Solve the 0/1 Knapsack Problem

Nº items	**Cx1**	**Cx2**	**CxU**	**TGA**
50	2610	2758	2887	2901
100	6642	6757	6807	7095
250	15736	15842	16005	16656
500	30870	31761	32356	33382

Table 3: Function Optimization: Summary of the Results (minimization)

		Genetic Operator											
		One-point Crossover			*Two-point Crossover*			*Uniform Crossover*			*Transformation*		
	Nº evals	**50000**	**100000**	**200000**	**50000**	**100000**	**200000**	**50000**	**100000**	**200000**	**50000**	**100000**	**200000**
Function	Rastrigin	88.170	88.170	88.170	67.639	67.639	67.639	63.739	63.739	63.739	73.272	52.518	**36.682**
	Griewangk	0.323	0.323	0.323	0.259	0.259	0.259	0.244	0.244	0.244	0.074	0.026	**0.010**
	Schwefel	665.406	665.406	665.406	557.770	557.770	557.770	456.273	456.273	456.273	220.878	62.404	**8.695**
	Ackley	16.248	16.248	16.248	15.102	15.102	15.102	14.181	14.181	14.181	11.617	8.645	**5.941**

4.2 RESULTS OBTAINED IN THE FUNCTION OPTIMIZATION DOMAIN

The TGA obtained, in the entire set of test functions, the best solutions after 200000 function evaluations. Table 3 reports the achieved results. The results presented are those obtained after 50000, 100000 and 200000 function evaluations using the SGA and the TGA. The best solutions are marked in bold.

In this case, the GA using the transformation mechanism evolves very slowly to the achieved result. On the other hand, the SGA converges very rapidly to the obtained value, but is unable to continue evolving. Besides, just like in the KP, in the function optimization domain, TGA obtained better results than SGA with fewer number of function evaluations.

The graphical representation shown in Figure 6 illustrates the SGA and TGA performances minimizing the Ackley function, but we observed a similar behavior in all the test functions. Once the population converges to a certain value, SGA is incapable of continue exploring other zones of the search space. The TGA evolves slower, but can continue improving during the 2000 generations.

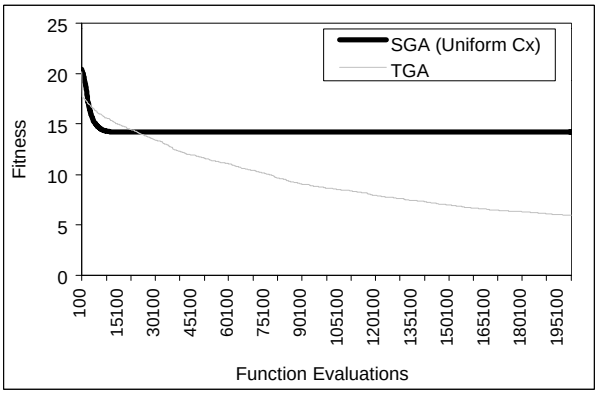


Figure 6: SGA and TGA evolution in 200000 function evaluations

Once again, these results appear to be a consequence of the loss of diversity in the population when using the crossover operators. TGA evolves during the entire simulation because the genetic variation of the individuals is kept in high levels. In the next section, we will focus the population's diversity measured in both problem domains.

Concerning the computational times, once again, TGA was the slower algorithm, but the differences to the times used by the crossover operators are quite small. TGA was approximately 7% slower than one-point crossover (the operator which obtained the worst results) and 4% slower than uniform crossover (which obtained the best performance among the crossover operators). Table 4 shows the times (in seconds) spent in the execution of the 25 trials for the minimization of the test functions.

Table 4: Time Spent to Minimize the Test Functions

Function	***Cx1***	***Cx2***	***CxU***	***TGA***
Rastrigin	7039	7059	7298	7698
Griewangk	5338	5360	5478	5686
Schwefel	4683	4722	4832	4989
Ackley	11320	11588	11667	12152

4.3 POPULATION'S DIVERSITY

The main reason for the good results obtained by the TGA seems to be the great diversity that the proposed mechanism introduces in the population. This can be the explanation for the fact of the TGA with 20 individuals outperforms the SGA with 200. To compare the diversity in the population we used a standard measure, which is the sum of the Hamming distances between all possible pairs in the population. This measure, when normalized, is defined as:

$$Div(Pop) = \frac{1}{LP(P-1)} \sum_{i=1}^{P} \sum_{j=1}^{P} HD(p_i, p_j) \qquad (11)$$

where L is the chromosome length, P is the population size; p_i is the i^{th} individual in the population and HD is the Hamming distance function.

Figure 7 shows the variation of the population's diversity for the KP. The results were obtained by the GA solving the KP problem with 100 items and compare the diversity

maintained by uniform crossover and transformation. To the other instances of the KP, the results were very similar.

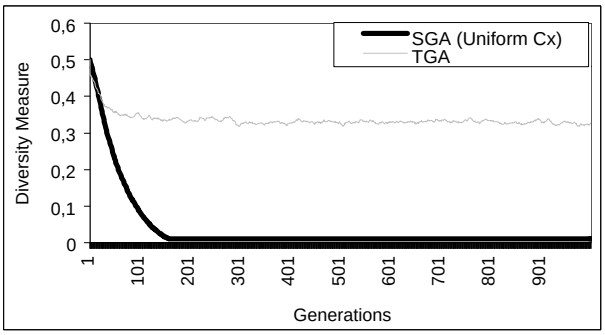


Figure 7: Population's Diversity in the KP

As we can see, the diversity of the population is higher when using transformation, indicating that the individuals are covering more areas of the search space. When applying uniform crossover, the population's diversity decreases to values near to zero avoiding the GA to continue evolving. In the Figure 4 we observed that the SGA stops evolving about generation 130. As Figure 7 indicates, the diversity of the population achieves the lower levels about generation 130.

In the domain of function optimization, the results were very similar. Figure 8 shows the diversity measure in the minimization of the Ackley function. Once again, there is a correspondence between the point where the diversity reaches low values and the point where the SGA stops evolving (20000 function evaluations in Figure 6).

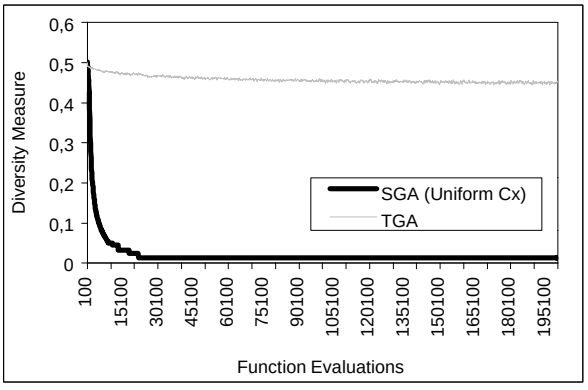


Figure 8: Population's Diversity minimizing Ackley Function

5 CONCLUSIONS

In this paper, we introduced a new genetic operator inspired in bacterial genetics, called transformation. We used this operator as an alternative to crossover and we studied the GA performance solving two different classes of problems. The results showed that the transformation mechanism is clearly superior to the SGA. Besides, with few individuals in population (or fewer function evaluations) transformation can achieve better solutions than crossover with larger populations.

Observing the population's diversity, we can see that transformation preserves a high degree of genetic variation among the individuals of the population.

We are currently using this genetic operator in a classical dynamic optimization problem and the preliminary results show that the TGA is able to adapt to the new solution when a change occurs (Simões and Costa, 2001b).

In order to enhance the GA performance when using this mechanism we are also implementing some modifications concerning some issues, namely, the assessment of the best transformation rate, the influence of the gene segment length and the generation of the gene segment pool.

Acknowledgements

This paper was partially supported by the Portuguese Ministry of Science and Technology under the program POSI.

References

W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone (1998). *Genetic Programming - An Introduction - On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann

D. Clark and L. Russell (1997). *Molecular Biology Made Simple and Fun.* Cache River Press.

I. De Falco, A. Iazzetta, E. Tarantino and A. Della Cioppa (2000). *On Biologically Inspired Mutations: The Translocation*. In Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference (GECCO'00), 70-77, Las Vegas, USA, 8-12 July 2000.

K. A. De Jong (1975). *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Dissertation, Department of Computer and Communication Science, University of Michigan.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc.

V. S. Gordon and D. Whitley (1993). *Serial and Parallel Genetic Algorithms as Function Optimizers*. In S. Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA5), 177-183, Morgan Kaufmann.

J. L. Gould and W. T. Keeton (1996). *Biological Science.* W. W. Norton & Company.

I. Harvey (1996). The Microbial Genetic Algorithm. Submitted to *Evolutionary Computation.* MIT Press.

J. H. Holland (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* 1st MIT Press edition, MIT Press.

Z. Michalewicz (1999). *Genetic Algorithms + Data Structures = Evolution Programs.* 3rd Edition Springer-Verlag.

M. Mitchell and S. Forrest (1994). Genetic Algorithms and Artificial Life. *Artificial Life* **1**(3):267-289.

N. Nawa, T. Furuhashi, T. Hashiyama and Y. Uchikawa (1999). *A Study of the Discovery of Relevant Fuzzy Rules Using Pseudo-Bacterial Genetic Algorithm.* IEEE Transactions on Industrial Electronics.

M. A. Potter and K. De Jong (1994). *A Cooperative Coevolutionary Approach to Function Optimization*. In the Proceedings of the Third Parallel Problem Solving from Nature (PPSN3), Jerusalem, Israel, 249-257, Springer-Verlag.

P. J. Russell (1998). *Genetics*. 5th edition, Addison-Wesley.

A. Simões and E. Costa (1999). *Transposition versus Crossover: An Empirical Study*. Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E. (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99), 612-619, Orlando, Florida USA, CA: Morgan Kaufmann.

A. Simões and E. Costa (2001a). *An Evolutionary Approach to the Zero/One Knapsack Problem: Testing Ideas from Biology.* In the Proceedings of the Fifth International Conference on Neural Networks and Genetic Algorithms (ICANNGA' 2001), 22-25 April, Prague, Czech Republic, Springer-Verlag.

A. Simões and E. Costa (2001b). *Using Biological Inspiration to Deal with Dynamic Environments*. Submitted to the 7th International Conference on Soft Computing (MENDEL' 2001).

Verification and Extension of the Theory of Global-Local Hybrids

Abhishek Sinha and **David E. Goldberg**
Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801, USA
{sinha,deg}@uiuc.edu

Abstract

This work is an extension of the framework for optimizing global-local hybrids. The existing theory idealizes the search problem as a search by a global searcher for acceptable targets or for basins of attractions which lead to acceptable target by invoking a local searcher. The two key parameters of this theory are—the probabilities of successfully hitting targets and basins and time-to-criterion values for different basins. First the existing theory is tested with variation in time-to-criterion values for the local searcher across several basins and is then extended to handle variations within individual basins. As a first step towards applying this theory to genetic algorithms (as the global searcher), selection dominated performance has also been studied in the context of this theory. The results are promising and make a strong case for further work in this direction.

1 INTRODUCTION

Past work has indicated that hybridization is a key factor for achieving superior performance with a genetic algorithm (GA) in many application domains. A pure GA can seldom match the performance of a method tailored to the problem at hand. A hybrid combines the global searcher (the GA) with other methods which exploit problem specific knowledge to generate better solutions than either could have come up with on its own. One of the issues central to hybridization is the efficient allocation of time between the global and local search. Most often the goals sought are to (a) maximize the reliability of reaching a solution of desired quality in a given amount of time or (b) minimize the time required to reach a solution of desired quality with given reliability. This study is towards developing efficient combines of global and local searchers to meet these goals.

The next section reviews some of the past work on hybrids and discusses the motivation for this work. This is followed by some theoretical background for tackling the problem at hand. Thereafter, some experiments to test the existing theory for variations in local time-to-criterion values across basins and then variations within the same basin are considered. The paper continues with the development of a model for selection (as a global searcher) and shows its significance in deciding between global and local searchers at different times. This is followed by suggestion of possible extensions. The study concludes with a brief summary and some comments on its significance.

2 PREVIOUS WORK

The applications literature of GA-local hybrids is too numerous to cite here, but the EnGENEous system (Powell, Tong, & Skolnick, 1989) was an early systematic hybrid of a GA and local search in a commercial setting. Davis (1991) was an early exponent of hybrids and his book gives a good rationale for so doing. Ibaraki (1997) describes the combination of optimization methods such as local search, dynamic programming and simulated annealing. with genetic algorithms for several combinatorial optimization problems.

Less has been said on the theory of global-local hybrids, but an important distinction between Baldwinian and Lamarckian learning was made by Hinton and Nowlan (1987). This issue of substituting the individual from the termination point of the local searcher into the population for further genetic search has been a much debated one. The study by Orvosh and Davis (1993) has interesting empirical results.

A majority of the work has been focused on narrow application domains (combinatorial optimization, Traveling Salesman Problem, etc.). A generic theoretical framework for combining GAs with other methods has been lacking.

The study by Goldberg and Voessner (1999) made a start towards addressing this issue by developing a framework for optimizing global-local hybrids. Preliminary results with random search as the global searcher and a quasi-Newton method as the local searcher were also published. This study builds on the above work. It verifies some of the extant theory and extends it for handling variation in time-to-criterion within a basin and of developed therein for application to GAs by tackling the first stage of a GA: selection.

In the next section we review necessary theory for this work.

3 BACKGROUND THEORY

This section is mainly drawn from other works (Goldberg & Voessner, 1999; Goldberg, 1991) and the interested reader is urged to refer to these papers for further details. A typical hybrid, H, consists of a global method G and a local method L. An iteration of H consists of one iteration of the G to generate a candidate solution which serves as the starting point for L which is invoked multiple times each consuming no more than an allowable time $\lambda_a : 0 \geq \lambda_a \leq \lambda_{max}$. This process continues until we exceed an allowable time T_a or the desired solution quality is obtained. The solution quality is the solution accuracy target $\phi \leq \phi_\tau$. In Figure 1, β_i (depicted as tessellated polygons) are the basins of attraction within which L can lead to the target solution which are depicted as islands τ_i.

The solution sought is better than some target value $\phi_\tau (= \phi^* + \Delta\phi$, where ϕ^* is the globally optimal maxima/minima and $\Delta\phi$ is the amount by which the sought solution quality differs from the ϕ^*). Now we consider the possible ways in which we can get to the target islands, τ_i. The union of the targets, the global region, $R_G = \bigcup_i \tau_i$. The probability of hitting R_G in a single invocation of the global searcher is denoted as P_G. For a random search with uniform distribution, P_G may be calculated by summing the areas of the targets and dividing by the total area of the search space.

The local time-to-criterion values λ_i are defined as the average number of time units required to get to the target starting from within the basin of attraction β_i. Although the time taken to reach the local optimum would depend on the exact point in the basin where we start from, here we consider a single λ_i over the whole basin for the sake of simplicity. The probability of hitting the basin β_i (exclusive of the target τ_i) with an invocation of G is denoted by P_i. Suppose G lands in a basin where the local search does not reach a solution of desired quality or in a basin where L fails to converge in $\lambda \leq \lambda_{max}$ time units. These regions are called *dead zones*. The probability of hitting the dead zone is denoted by P_D and can be calculated as follows

$$P_D = 1 - P_G - \sum_i P_i. \tag{1}$$

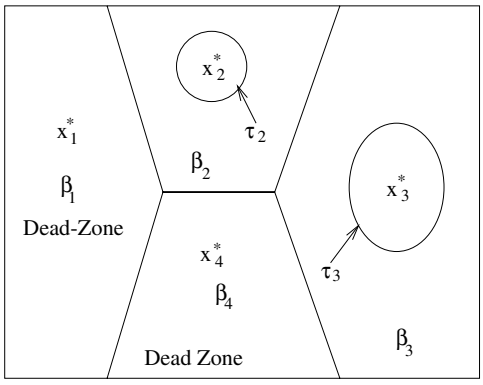


Figure 1: A 2-D sketch of the search space showing the target islands, τ_i, basin of attraction under local method L to those targets, β_i, and dead zones.

The global search is assumed to take one unit time and the local search times are calculated relative to that. Calling the allowable local time constant λ_a, and the average local time constant $\overline{\lambda}$, the solution time T consumed in n global-local iterations is:

$$T = (1 + \overline{\lambda})n. \tag{2}$$

Some of the basins may have a local time-to-criterion higher than the allowable time for local search, λ_a. Hence the probability of hitting the global zone can be found by summing the probability of hitting global region initially (by the global searcher) and the probability of hitting the basins with a time-to-criterion less than the allowable, λ_a. This can be stated as

$$P_{\lambda_a} = P_G + \sum_{i:\tau_i \neq 0, \lambda_i \leq \lambda_a} P_i. \tag{3}$$

Next, we have the formulation for minimizing time for certain reliability.

3.1 MINIMUM TIME FORMULATION

The probabilistic error, α_a, is defined as the probability of not reaching a solution of desired quality. For

a specified allowable error α_a, the reliability condition can be written as

$$\alpha_a = (1 - P_{\lambda_a})^n, \tag{4}$$

where n is the number of iterations. By eliminating n and minimizing, we get:

$$\min(\lambda_a + 1)\frac{\ln \alpha_a}{\ln(1 - P_{\lambda_a})}. \tag{5}$$

This gives us the minimum time required to reach a solution with a specified allowable error α_a. Next, we show formulation for maximum reliability in a given time.

3.2 MAXIMUM RELIABILITY FORMULATION

We have $n = \frac{T_a}{\lambda_a + 1}$. The maximum allowable time $\lambda_{max} \leq T_a - 1$. Minimizing the error and substituting for n gives

$$\min(1 - P_{\lambda_a})^{\frac{T_a}{\lambda_a + 1}}. \tag{6}$$

We should go with G alone when $(1 - P_G)^{T_a} < [1 - (1 - P_D)]^{T_a/\lambda'_a}$ (Goldberg & Voessner, 1999) which can be reduced to

$$P_D > (1 - P_G)^{\lambda'_a}, \tag{7}$$

where $\lambda'_a = \lambda_a + 1$.

Next, we report some experiments which verify different aspects of the theory on two different test functions.

4 EXPERIMENTS

This theory has been verified using random search a G and a quasi-Newton, the Broyden Fletcher Goldfarb Shanno (BFGS) method (Press, Teukolsky, Saul, Vetterling, & Flannery, 1992) as L for uniform λ across several basins (Goldberg & Voessner, 1999). The BFGS method has a useful property of having nearly equal convergence times across geometrically similar basins and the time taken is nearly the same irrespective of the starting point of the search within the basin. The test function used in that work was:

$$f(x, y) = \begin{cases} \frac{d_i}{r_i^2}(\overline{r}^2)(2 - \frac{\overline{r}^2}{r_i^2}) - d_i & \text{for } \overline{r}^2 \leq r_i^2 \\ 0 & \text{otherwise} \end{cases}$$

where $\overline{x} = x - cx_i$, $\overline{y} = y - cy_i$, $\overline{r}^2 = \overline{x}^2 + \overline{y}^2$, and $c_i = \{(2.0, 8.0), (3.0, 4.0), (5.0, 7.0), (7.0, 8.5), (7.0, 4.0)\}$, $r_i = \{1.5, 2.0, 0.5, 1.0, 2.5\}$, $d_i = \{2.0, 3.0, 2.0, 4.0, 2.0\}$.

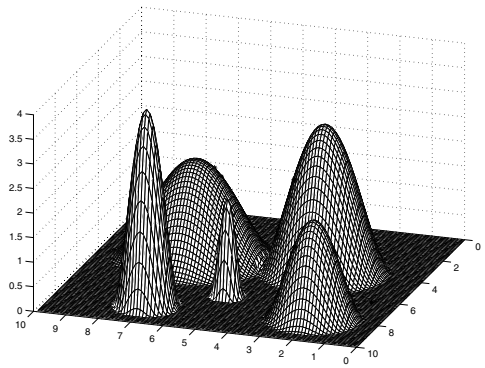


Figure 2: The inverted test function (-f(x,y)) with five quasi concave basins.

The global minima is -4.0 and is located at $(7.0, 8.5)$. Figure 2 shows the function. This function has been used for all of the following experiments except where mentioned otherwise. Similarly G is a random search with uniform distribution except where mentioned otherwise. L is the BFGS method throughout.

The termination criterion for all simulations was a maximum error of 0.01%. For random search P_G is calculated by summing the areas of the targets and dividing by the total area of the space. A Baldwinian approach is followed wherein the result from the local searcher is not substituted back into the original population, but the value found by local search is used to evaluate the starting point.

There are two possible ways to measure time in these experiments. One is clock or execution time of the block of code representing the global and local searchers. One problem with this approach is the lack of high resolution timers for most platforms. Also keeping track of actual execution time at different points is cumbersome. Another way is to assign appropriate weights to function and derivative evaluations and use this computation as a measure of time. In real-world applications function evaluations tend to be the bottleneck. Also it is much easier and convenient to track these rather than the exact time. Keeping this in mind, the latter approach has been pursued in this work.

4.1 CASE I: VARIATION IN λ_i

With different λ_i for different basins choosing an appropriate value for λ_a becomes critical. The possible choices for λ_a are the different λ_i. A higher λ_a is appropriate if the cumulative probability of success increases sufficiently. The following procedure is

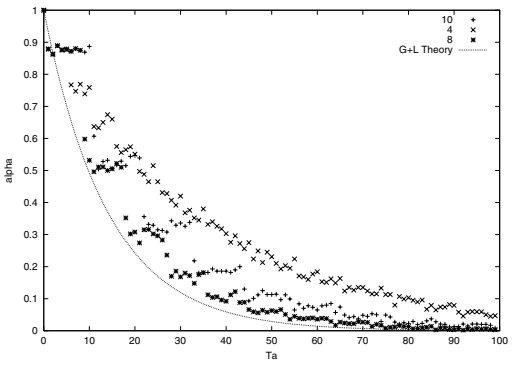


Figure 3: The probabilistic error α is shown as a function of the allowable time T_a with different λ values for the local searcher L. The lowest error is obtained for $\lambda_a = 8.0$, as suggested by the theory.

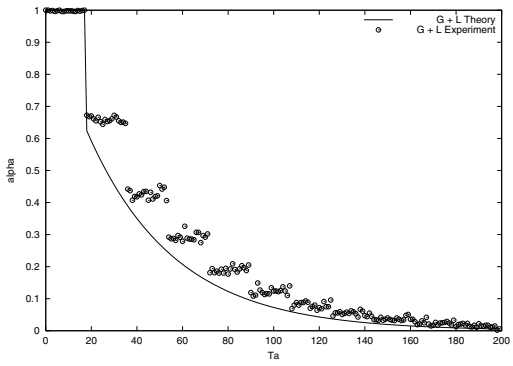


Figure 4: The probabilistic error α is shown as a function of the allowable time T_a for $\phi_\tau = 0.02, P_G = 0.002$. In this case $G + L$ yields lower error.

adopted for choosing an optimal value of λ_a. First, the basins are arranged in ascending order of λ values. Locally optimal choices are obtained by comparing the λ values of the ith basin and the $(i+1)$th basin on the basis of probabilistic error. The error is given by equation 4 with $n = T_a/(\lambda_a + 1)$:

$$\begin{aligned}\alpha_a &= (1 - P_{\lambda_a})^{T_a/(\lambda_a+1)}, \\ &= P_D^{T_a/(\lambda_a+1)}.\end{aligned}$$

The locally optimal choice with the least error gives the globally optimal choice of λ_a. P_D is calculated theoretically. This procedure yields $\lambda_a = 8.0$ as the optimum value for the aforementioned function.

For this experiment we assign different λ_i values to different basins. The assignment was $\lambda_i = \{6.0, 12.0, 4.0, 10.0, 8.0\}$. Whenever the global searcher lands in a basin, it is assumed that the local searcher takes the assigned amount of time to reach a target. The desired solution quality was $\phi_\tau = -1.0$. Figure 3 shows the results for this case. The least error is with $\lambda_a = 8$, which was predicted by the theory. For sake of clarity , only three of the five possible choices of λ_a have been plotted. The other choices of λ_a (which are not shown) also led to inferior performance.

4.2 CASE II: VARIATION OF λ WITHIN A BASIN

For most real-world functions the time-to-criterion values depend upon the starting point of the local searcher. This case is illustrated by the Griewank function (Törn & Žilinskas, 1989) (for the two-dimensional case)

$$f(x_1, x_2) = 1 + \frac{{x_1}^2 + {x_2}^2}{4000} - \cos(x_1)\cos(\frac{x_2}{\sqrt{2}}) \quad (8)$$

$x_1, x_2 \epsilon [-512, 511]$. This is a differentiable, multimodal function with the global minima as 0.0 located at $x_1 = 0.0$ and $x_2 = 0.0$.

For a solution quality of $\phi_\tau = 1.5$ λ varied from 3.0 to 24 for each of the basins. For the sake of convenience in calculating the probabilities only a portion (with 4 basins including the one with the global minima) of the space was considered. The area of each basin was approximated by a circle. To calculating P_i, λ was chosen so that 90% of the observed λ were below this value. The above procedure reduced the effective area of the basin by 10%. This was factored into the calculation of P_i. Since the behavior was similar across basins, λ_i were taken to be uniform. Figure 4 shows the results for a desired solution quality $\phi_\tau = 0.02$. The results are according to the theory. Here $P_G = 0.002, P_D = 0.6237, \lambda_a = 17.0$. The global searcher cannot succeed without the help of L. Figure 5 shows the results for a desired solution quality $\phi_\tau = 1.5$. For this case, $P_G = 0.356, P_D = 0.55, \lambda_a = 3.0$. With a higher P_G here we see that G alone performs better than $G + L$ combined. According to the theory one should proceed with G only if $P_D > (1 - P_G)^{\lambda'}$ which is the case here. The theory holds for this case.

In the foregoing experiments, G has been taken as uniform random search, making the correspondence between theory and experiment quite close. As we move away from random search and toward genetic and evolutionary algorithms as our choice of G, we consider the changes necessary in the theory to accommodate the more complex global search. The next section takes our first steps in these directions by consider-

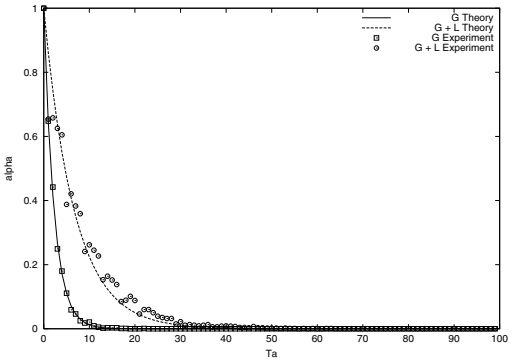


Figure 5: The probabilistic error α is shown as a function of the allowable time T_a for $\phi_\tau = 1.5, P_G = 0.356$. In this case G alone performs better.

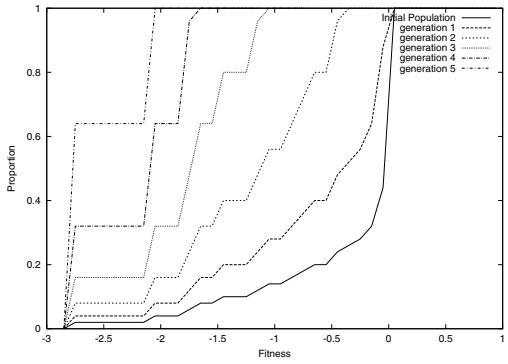


Figure 6: Proportion of population above different fitness levels is shown for different generations with truncation selection.

ing *selection dominated performance*. This will result in modifications to the basin probabilities P_i as the generations progress.

5 TOWARDS $G = GA$: THE ASSUMPTION OF SELECTION DOMINATED PERFORMANCE

A typical GA can be decomposed into the following steps: (a) random initialization, (b) selection, (c) crossover and (d) mutation. So the next logical step is to incorporate selection into the theory. Selection tends to dominate early GA performance and proceeding with selection alone as a choice for G would be a positive step toward having GAs as the global searcher. Some of the ideas in this section are drawn from (Goldberg, 1991).

First we develop a model for selection which enables the prediction of the population fitness level in successive generations. This model also enables the calculation of probabilities of reaching a solution of desired quality using G alone. This information can then be used for choosing between G and $G + L$. We verify these ideas with some experiments.

Truncation selection was used for modeling selection as it lends itself to easier modeling. Figure 6 shows the proportion of individuals above various fitness levels at different generations under selection. Using truncation selection we can approximate the proportion of individuals below a certain fitness level in successive generations with a power law as follows:

$$f_{norm} = (f - f_{min})/(f_{max} - f_{min}), \tag{9}$$

$$p = (1/s)^t = f_{norm}^b. \tag{10}$$

f_{norm} is the normalized fitness, p is the proportion of the population above a certain fitness level, f, t is the generation number, s is the selection pressure and b is a constant. Figure 7 shows the approximated curve with $b = 5.376$ and the proportion curve for the random initialization of the population (i.e. $t = 0$).

Solving for f_{norm} and substituting $s = 2$, we have

$$f_{norm} = (1/2)^{t/b}, \tag{11}$$

$$f = f_{norm}(f_{max} - f_{min}) + f_{min}. \tag{12}$$

This gives us the fitness level above which all the individuals in the population lie for any generation t. This may be used to obtain a closed form solution for the proportion of population above a certain fitness level as it is shown. After the initialization the individuals are randomly distributed over the whole search space. During selection s copies are given to each of top $1/s$ proportion of the population. This leads to multiple copies of the fitter individuals, but their distribution remains random in space. So the probability P_G of reaching a point better than the current level of the population, f, may be calculated by summing the areas for the desired fitness across all basins and dividing by the summation of the areas at current fitness level over all the basins. If the current fitness level, f, happens to be better than the desired fitness then all the individuals in the population meet the criteria and $P_G = 1$.

Since selection alone does not provide any new points, if we set a very high solution quality criterion selection fails to reach the criterion all together because the probability of having such a point after a random initialization would be very low. But if the criterion is within reach (i.e. already present in the initial population) of the selection process the convergence is very

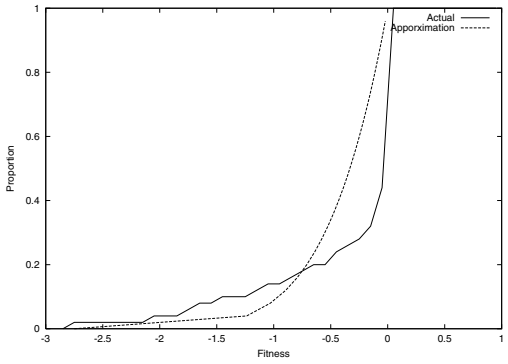


Figure 7: The proportion of individuals above certain fitness levels modeled by a power law with $b = 5.376$.

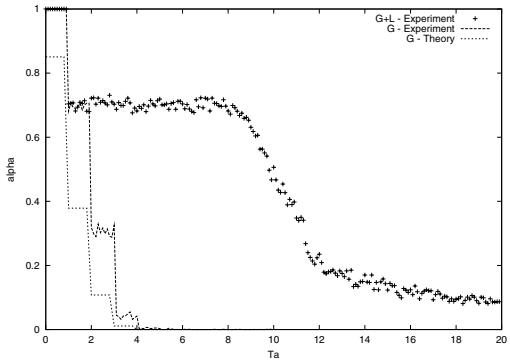


Figure 8: The probabilistic error α is shown as a function of the allowable time T_a for $\phi_\tau = -1.0$ with selection as the global searcher. G alone yields lower error.

fast. In this case selection alone fairs better than a combination of selection and local search. The high relative cost of the local search proves to be an overhead. But when none of the initial points meet the criteria, the local searcher plays a key role in leading to an acceptable solution.

5.1 EXPERIMENT

Here selection was used as the global searcher in conjunction with BFGS. A population of 50 individuals was generated randomly. At each iteration of G one truncation selection ($s = 2$) was carried out and one randomly chosen individual from the resulting population was returned. The local search then took over with this individual as the starting point. This was a single iteration of $G + L$. The output of the local search was not substituted back into the population. The results from the aforementioned model were used to calculate theoretical P_G values. These values were no longer constant after each iteration (as was the case with random search). Figure 8 shows the results. The experimental results for G are in close agreement with the one derived from the model. The difference can be attributed to approximation errors in the model. G alone performs better for this case. The theory for $G + L$ requires more work and will be addressed in a later study.

6 EXTENSIONS

This work has initiated a first step towards applying global-local hybrid theory to GAs. A number of extensions suggest themselves:

1. Consider selection dominated performance theory with real GAs with crossover and mutation.

2. Currently, the theory uses stationary calculation of probabilities. Modify theory to handle non-stationary probabilities.

3. Test the theory on a rigorous test suite.

4. Consider off-line and online methods to determine theory parameters for real problems.

5. Consider extension to more than 2 methods.

The above steps can provide answers to some important questions: When should theory be expected to be good? What are the dimensions of hybrid difficulty? How should one estimate theory parameters in the absence of complete knowledge of the fitness landscape? As practitioners seek answers to these questions, researchers will soon direct their efforts to explore these directions.

7 SUMMARY & CONCLUSIONS

Global-local hybrid theory is increasingly being shown to be a useful tool for dividing labor between multiple solution techniques to obtain quality solutions efficiently. When G is random search, the theory may be used to choose efficient combines with local searcher, L, resulting in λ variation both between and within basins of attraction. When G is not a random search, it appears that modifications can be made to design efficient combinations. By considering selection dominated performance, a preliminary extension of the theory towards GAs was proposed. More work is needed, but these results and extensions promise a practical design capability for efficient hybridization.

ACKNOWLEDGMENTS

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0163. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.

Thanks are also due to Kumara Sastry and Martin Pelikan for comments on a draft of this paper.

References

Davis, L. (1991). *The handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems*, *5*(2), 139–167.

Goldberg, D. E., & Voessner, S. (1999). Optimizing global-local search hybrids. *Proceedings of the Genetic and Evolutionary Computation Conference - 1999*, 220–228.

Hinton, G. E., & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, *1*(1), 495–502.

Ibaraki, T. (1997). Combinations with other optimization methods. In Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation* (Section D3). New York: Oxford University Press.

Orvosh, D., & Davis, L. (1993). Shall we repair? genetic algorithms, combinatorial optimization, and feasibility constraints. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 650.

Powell, D., Tong, S. S., & Skolnick, M. M. (1989). Engeneous domain independent, machine learning for design optimization. *Proceedings of the Third International conference on Genetic Algorithms*, 151–159.

Press, W. H., Teukolsky, S. A., Saul, A., Vetterling, W. T., & Flannery, B. (1992). *Numerical recipes in C* (2nd ed.). Cambridge , MA: Cambridge University Press.

Törn, A., & Žilinskas, A. (1989). *Global optimization*. Berlin: Springer.

Modelling Gas with Self Adaptive Mutation Rates.

Jim Smith
Intelligent Computer Systems Centre
University of the West of England
Bristol, U.K.
www.csm.uwe.ac.uk/~ jsmith
James.Smith@uwe.ac.uk

Abstract

This paper introduces a dynamical systems model of a generational Genetic Algorithm with Self-Adaptation of mutation rates. This model is used to predict the mean fitness of an evolving population as a function of time. The accuracy of these predictions are then tested by running a series of experiments using Genetic Algorithms with different population sizes. It is shown that although there is a threshold below which the "real" populations do not closely follow the predictions, the model is still able to give us useful information about the behaviour of the "real" GAs, since the populations tend to get "stuck" at points close to certain eigenvectors of the infinite population model. Arguments are given which allow the prediction of which eigenvectors will be important.

The dynamics of the population evolving on a non-stationary environment are then considered, and some conclusions drawn about the nature of environmental change to which the algorithm will be able to respond.

1. Introduction

This paper applies the dynamical systems models of Genetic Algorithms (GAs) developed by Vose [9], to a simple model of a genetic algorithm with self adaptive mutation rates. In particular the behaviour of the algorithm is considered on problems of unitation, where the approaches of [8] [5] can be taken to reduce the dimensionality of the dynamical systems to be solved. Although these are infinite population models, it has been shown that provided the population size is sufficiently greater than the number of equivalence classes considered, then the models can accurately predict the behaviour of "real" GAs.

The Self Adaptation of mutation rates within a GA was first proposed by Bäck in [1], who used a binary encoding for the mutation rate within a generational GA. This idea was expanded by Smith and Fogarty [6] who examined a number of different encodings within the context of a Steady State GA, and Hinterding [3] who used a real number encoding for mutation step sizes to act on (effectively) real-valued genes.

In this model a generational model is used similar to that of [1]: an individual is deemed to have a single mutation rate, m attached to it, which takes one of a fixed number, q, of values. Mutation is a two phase process, where first the value of m is varied to yield a new value m', then the problem representation is mutated with this new bit-wise mutation probability. For the purposes of clarity, we will restrict ourselves here to the situation where in the first process a new value $m \in \{1, \ldots, q\}$ is chosen at random with probability z, and the value is left unchanged with probability $(1-z)$. We will refer to z as the Innovation Rate. Note that by setting $q = 1$, we can use this model for fixed mutation rates as well. In[4] the results are given of experiments which demonstrate that the mechanism described here is sufficient to permit adaptation of the mutation rates to optimal values on a range on NK landscapes with $N = 40$ and $K \in \{0, 15\}$.

This model represents a simplification of the algorithms described in [1, 6], in two aspects. The first of these is that rather than using a binary (or Gray code) encoding for the mutation rate,(which is itself subject to bitwise mutation) the mutation rate is represented by a single allele of alphabet q. In this work q is taken to have a much smaller value (10) than the number of values considered by Bäck and Smith, so as to render the resultant matrices more tractable, although this is not a necessary restriction. However there remain

implications for the inheritance of mutation "genes", since these were subject to recombination in the works mentioned above and are not in this system.

The second simplification concerns the way in which the mutation rates are themselves subject to change. In [1, 6] the binary strings representing the mutation rates are first decoded, then are themselves subject to bit-wise mutation at the decoded rate. This has the effect that (subject to decoding effects such as"Hamming cliffs") mutation rates are likely to change to similar values. By contrast, in this model rates are changed to a randomly selected value with probability z, and in fact the value is *altered* with probability $z*(q-1)/q$ since the same value may be chosen. This decision was made so as to simplify the derivation and explanation of the mixing matrix, and (more importantly) because experimental results showed better ability to adapt to changing environments than systems where the rate was more likely to change to a similar value. However it is a fairly trivial matter to change the mixing matrix to a "similarity" based adaptation, once a set of suitable probability distributions has been chosen. The main factor would be the explicit choice of what probability distribution to use for each value, as opposed to the implicit choices made when a bit string (whether binary or gray coded) is used.

In this paper we will consider problems of unitation, i.e. where the fitness of an individual solution depends solely on the number of 1's in its binary representation. For such a problem with a representation of length l there are $l+1$ equivalence classes of solutions with different representations but equal fitnesses i.e. for a three bit representation the four classes are:

{000}, {100, 010, 001}, {011, 101, 110} , {111}.

In order to deal with the different mutation rates we will extend this model so that each of the fitness equivalence classes is subdivided into q further classes according to the mutation rate attached to the individual solutions, giving a total of $N = q*(l+1)$ states. For an individual state i the fitness is $f'(i) = f(i/q)$ and the mutation rate is indexed by $m = i\%q$, where the % symbol has its usual modulus meaning, and the division i/q is taken to be rounded down to an integer value.

We can therefore define a population vector $\overline{p} = (p_1, \ldots, p_N)$ such that the components p_i represent the proportions of the population in class i, subject to the restriction $\Sigma_i p_i = 1$.

Following Vose's model, we can model the effect of the GA on this population vector as

$$\overline{p'} = G\overline{p} = MF\overline{p} \tag{1}$$

where the functions M and F represent the mixing (mutation and crossover) and selection operators respectively. The outcome, $\overline{p'}$ represents the probability distribution from which the next population will be sampled, which is equivalent to the next generation in the *Infinite Population Model* (the reader is referred to [9] for a more detailed discussion).

For fitness proportional selection, the selection operator can be modelled by using a diagonal matrix S with elements

$$S_{ij} = \begin{cases} f(i/q) & i = j \\ 0 & i \neq j \end{cases} \tag{2}$$

and the operation of the selection operator is given by

$$Fp = \frac{Sp}{\langle f \rangle(p)} \tag{3}$$

where

$$\langle f \rangle(p) = \sum_{j=0}^{N} p_j f(j/q) \tag{4}$$

Crutchfield et al have shown that this can be turned into a linear form, and derived equations for the calculation of the mean fitness as a function of time without the need for iterated matrix multiplication [8]. Although both methods were used (for the purposes of testing), in practice it was found that the "brute force" method could be executed in reasonable time on a 500MHz Pentium III, using code developed in C using the "meschach" libraries [7]. As an indication of the roundoff errors, the eigenvector corresponding to the maximum eigenvalue for the One-Max problem, which should be positive, was computed to have a single negative component of size $< 1.1-^{-17}$.

The Mixing matrix, M can be further decomposed into two functions representing crossover and mutation, a derivation in this case will be given in the next section. Given a form for M and F we can revisit (1) and consider the case for fixed points (if they exist) of the algorithm. If v is a fixed point of the system (i.e. $GV = v$) then we have:

$$MSv = \langle f \rangle(v)v \tag{5}$$

i.e. v is an eigenvector of MS with eigenvalue equal to its average fitness. Theory tells us that there will only be one eigenvector of the system corresponding to a "real population" (i.e. which lies within the simplex), and that this corresponds to the population whose

mean fitness is equal to the biggest eigenvalue. However, other authors have shown eigenvectors of the system which are close to the simplex, (i.e. which are almost attainable by real populations) can act as attractors for evolving populations in "real" GAs [5]. Given the increased number of states of the system studied here, it can be expected that more eigenstates with similar eigenvalues will be present. The hypothesis is that analysis of the composition of the eigenvectors will inform our understanding of the adaptive process.

2. Derivation of Mutation Matrix

In the derivations below we will use the following notation:
$P_s(i) = Sp_i$ denotes the probability of selecting an individual of class i.
$P_r(ijk)$ denotes the probability of generating a member of class i by recombination between a member of class j and one of class k.
$P_m(ij)$ denotes the probability of creating a member of class i by mutation from one of class j

Assuming the regular order of selection-crossover-mutation, we know that the i^{th} component of the probability distribution vector for the next generation, is simply the sum over all classes j, of the probability of generating a member of class j from the selection-crossover process, multiplied by the probability of mutating that individual from class j to class i, i.e.

$$Gp_i = \sum_{j=1}^{N} Pm_{ij} * \sum_k \sum_l P_r(jkl) * P_s(k) * P_s(l) \quad (6)$$

Initially we will restrict ourselves to the case where there is no crossover, and the first parent is simply copied, i.e.

$$P_r(ijk) = \begin{cases} 1.0 & i = j \\ 0.0 & i \neq j \end{cases} \quad (7)$$

and only one of the P_s terms is used. We can further simplify (6) by noting that we can separate the summation into two parts. In the first of these, the member selected to be copied has the same attached mutation probability as i (i.e. $i\%q = j\%q$) , and this is unchanged. In the second, the attached mutation rate is different, but with probability z this is changed, achieving the correct rate with probability $1/q$.

Common to both of these is the fact that the problem representation must then be mutated to contain the same number of 1's. If we use the notation that a mutation rate of m_i is attached to the class i, a of the j/q ones get mutated to zero and b zeroes get mutated to one, this happens with probability :

$$P'_m(ij) = \sum_{a=0}^{j/q} \sum_{b=0}^{l-j/q} \delta_{j/q-a+b,i/q} * \binom{l-j/q}{b} \binom{j/q}{a} m_i^{a+b}(1-m_i)^{l-a-b} \quad (8)$$

where $\delta_{x,y}$ is the Kronecker delta function:

$$\delta_{x,y} = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases} \quad (9)$$

Taking the two parts together gives us:

$$\begin{aligned} Gp_i &= (1-z) * \delta_{i\%q,j\%q} \sum_j P_s(j) * P'_m(ij) \\ &\quad + \frac{z}{q} * \sum_j P_s(j) * P'_m(ij) \quad (10) \\ &= \sum_{j=0}^{N} P_s(j) * P'_m(ij) * \left(\frac{z}{q} + \delta_{i\%q,j\%q}(1-z) \right) \end{aligned}$$

from which we can see by inspection and comparison with (1) that the elements of the mixing matrix M are:

$$M_{ij} = \left(\frac{z}{q} + \delta_{i\%q,j\%q}(1-z) \right) * P'_m(ij) \quad (11)$$

where $P'_m(ij)$ is defined as per (8).

3. An Example: One-Max

In order to compare the predictions of this model with the performance of "real" GAs - that is to say ones with finite populations- a series of experiments were made using the "One-Max" function. A problem length (L) of fifty bits was used, along with ten different mutation rates (i.e. $q = 10$), yielding a system with 510 equivalence classes. The mutation rates used were from the set {0.0005, 0.001,0.0025, 0.005, 0.0075, 0.01,0.025, 0.05, 0.075, 0.1}, and the rate attached to an individual was changed to a new random value with probability $z = 0.01$. In practice it was found that because of the number of classes with selection probability 0.0, a singular matrix was generated, and so the fitness function was modified by adding 1.0 to each (scaled) value, giving fitness values in the range $1-101$. The model was used to predict the mean population fitness as a function of time, the eigenvectors of the system, their corresponding eigenvalues, and a number of metrics relating to the distance of those eigenvectors from the simplex. A series of experiments

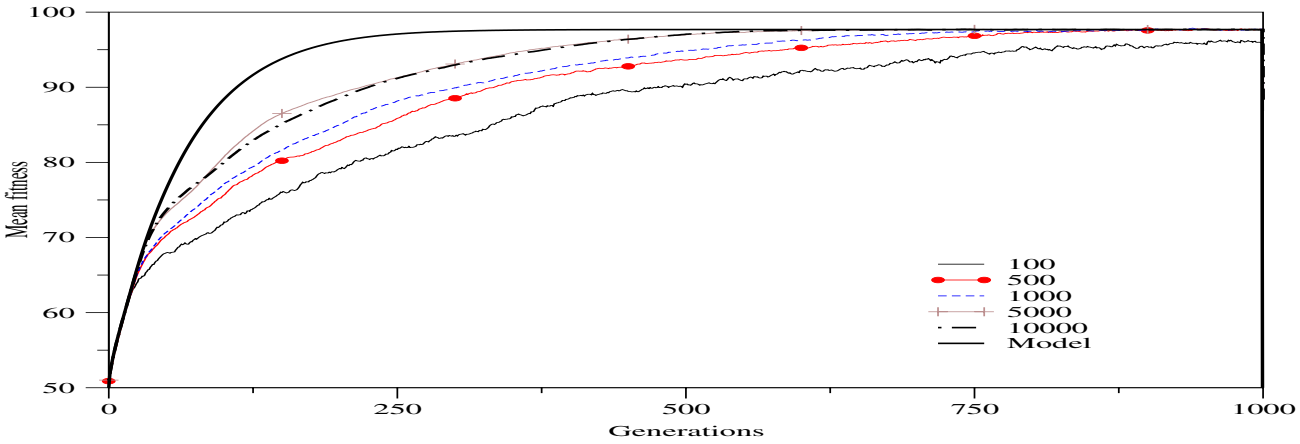


Figure 1: Evolution of Mean Fitness: Predicted vs. Empirical Results

were then run using a Self Adaptive GA with the same mutation values, no crossover, and a range of different population sizes. Baker's SUS selection algorithm [2] was used rather than "Roulette wheel" as it has been shown to exhibit less noise, a factor which becomes more important as we move from an infinite population model to the vagaries of stochastic effects with finite populations. Figure 1 shows the predicted trajectory of the population's mean fitness against time, along with empirical results for different population sizes, averaged over twenty runs of the GA.

As can be seen, the experimental curves deviate from the predicted values, by an amount that decreases as the population size increases. Allowing the GA to run for longer periods showed that the mean fitness did converge onto the predicted value, regardless of population size.

In [5] results are reported for a GA on this problem with $L = 20$ and a population of 500, i.e. at least an order of magnitude larger than the number of equivalence classes, and in [8] a ratio of population size (μ) to number of classes $\mu > 2^N$ is used in order to obtain results that match the predictions. For the self-adaptive GA, this presents a problem, since the number of equivalence classes in our model is increased by a factor q.

As can be seen from Figure 1, population sizes of greater than 1000 are needed to obtain a close match with predictions, although for all population sizes there is a good match in the first stages of the evolution. The need for large population sizes can be easily understood by the fact that in a "real" GA, the population vector is effectively discretised with a scale factor of $1/popsize$.

In the model, the population is able to sustain initially small proportions of individuals falling into the highest fitness classes, which are created either through random initialisation, or (with very low probability) via the mutation of less fit individuals. These will then increase exponentially according to their fitness relative to the population mean.

With finite populations, the discretisation effect is such that very few, if any, of these individuals are created, and the population is consigned to a more gradual evolution of fitness.

However, the deviations from the predictions caused by stochastic effects do not mean that the model is of no use. In Figure 2 the mean fitness for a single run with a population size of 2000 is shown, along with the Euclidean distance (multiplied by 1000) to the nearest eigenvector, and the fitness of that eigenvector (i.e. it's eigenvalue). The epochal nature of the search can clearly be seen, with the population being attracted to a succession of increasingly fit eigenstates of the system, before converging around the stable eigenvector. Note that convergence in this case is in terms of phenotypic fitness rather than genotypes, since many genotypes will belong to the same equivalence class as discussed above.

In [8] the systems studied contain a small number of equivalence classes, and the empirical results show that the populations do spend "epochs" at the mean fitness levels corresponding to the eigenvectors of the system. Arguments are developed to explain the differing amounts of time typically spent at each fitness level. However for the systems studied here, there are much larger numbers of eigenvectors of the system, and the problem becomes one of predicting which of these will become attractors for the population as it evolves.

Figure 3 shows the eigenvalues (mean population fit-

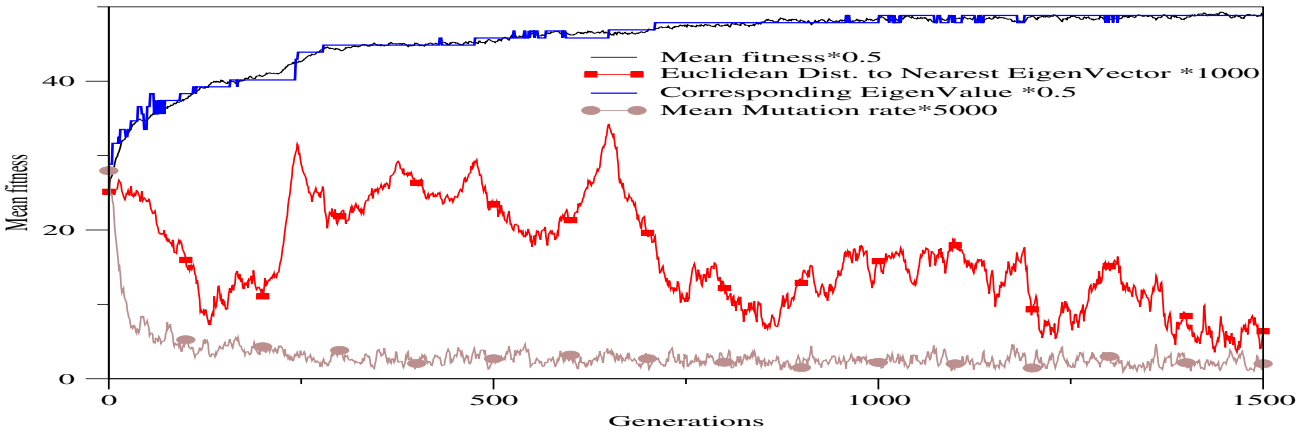


Figure 2: Population Dynamics for a single run with population size 2000. The population mean fitness, distance of population from current nearest eigenvector (x1000), and value of corresponding eigenvalue are shown.

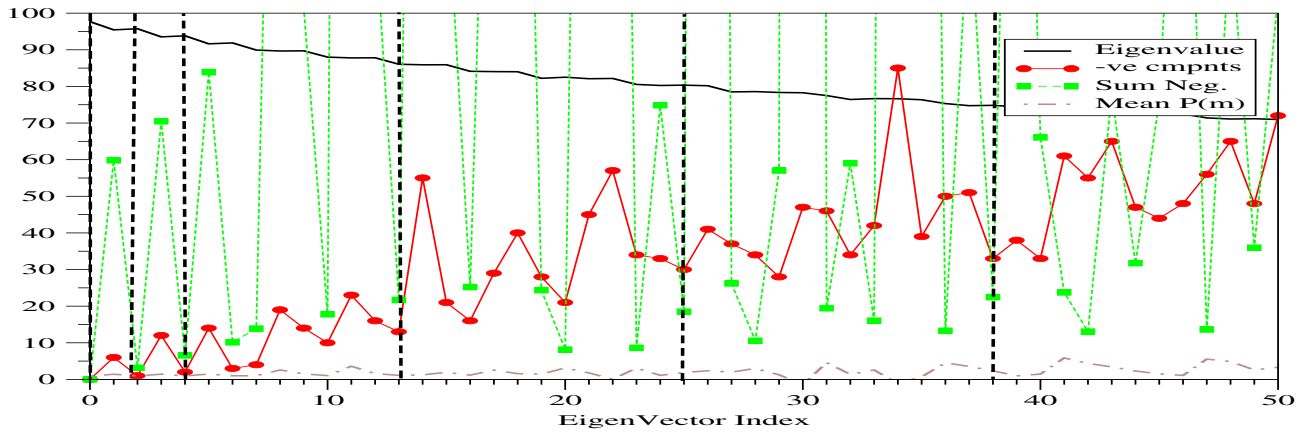


Figure 3: Analysis of Eigenvectors, showing corresponding eigenvalue, number of negative components, sum of negative components, and mean mutation rate the last two scaled for visualisation. The dashed vertical lines represent the fitnesses of the epochs seen in the previous figure

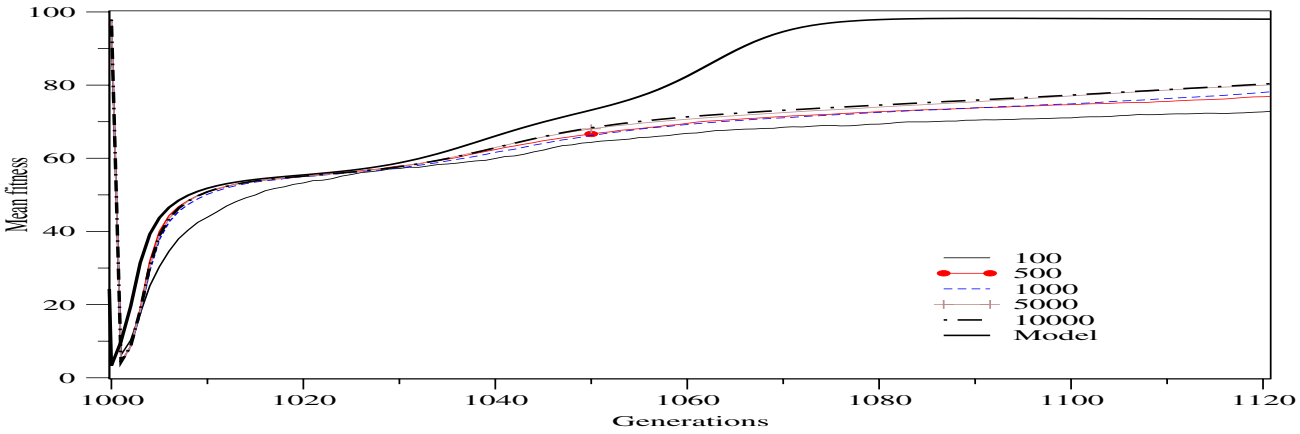


Figure 4: Evolution of Mean Fitness After Environmental Change:Predicted vs. Empirical Results

nesses) of the first fifty eigenvectors of the system, along with the number of negative components, their sum, and the mean mutation rate of a population at the eigenvector. Also marked by vertical dashed lines are the epochs noted in Figure 2. As can be seen these correspond to eigenvectors of the system which are a) close to the simplex, b) have few negative components and c) have a range of mutation classes present. Note that the first two points above although related are not linearly related. Point c) is indicated by the fact that the mutation rate fluctuates between extremes, but the epochs occur between these fluctuations, and this has been confirmed by closer analysis of the experimental log files. Analysis of subsequent runs with different population sizes showed that epochs always occurred close to eigenvectors with the properties listed above.

4 Adaption in Dynamic Environments

In order to begin to understand the ability of the algorithm to adapt to dynamic environments, it is first necessary to compare the predicted and observed behaviour of the algorithm in response to transitions in dynamic environments. This was achieved by running experiments as above, but extended to 2000 generations, with a change after 1000 generations from One-Max to Zero-Max.

For the GAs, this change was trivial to implement. For the model, this could have been achieved by changing the values in the selection matrix, and the recalculating G and its eigensystems. In practice it was easier to utilise the symmetry of the two problems, and leave the matrices as above, and assume that prior to the change the population vector would have converged onto the stable eigenvector for the One-Max problem. This vector was then translated into the equivalent for the Zero-Max problem according to:

$$p_i^0 = p^1_{((L-i/q)*q+i\%q)} \tag{12}$$

where the superscripts indicate the problem(Zero-Max or One-Max).

Figure 4 illustrates the results of these experiments, concentrating on the period immediately after the transition. As can be seen there is a good match between the prediction and the observed behaviours, especially in the first thirty or so generations.

The patterns of (predicted and observed) evolved behaviour starting from a converged population are very different to those with the initial random population. This can be explained by examining the proportions of the population falling into the different mutation classes as shown in Figures 5 (predicted) and 6 (observed). Initially after the transition, mutations are on average beneficial, and so once re-introduced by chance, individuals with the highest mutation rates attached start to take over the population. However once the mean fitness has passed 50%, then on average mutations will be deleterious, and so there is a phase transition, and individuals with lower mutation rates attached have a selective advantage. Inspection of Figures 5 and 4 shows that this happens around 25 - 50 generations after the change. It is notable that the empirical and theoretical results are virtually identical up to this point for all population sizes over 100. From Figures 5 and 6 it can be seen that with a finite population the lowest mutation class does not takeover the population as much as is predicted after the phase transition. It has been suggested above that this is because the model predicts the early generation of individuals with high fitness, for which low mutation rates are selective advantageous, whereas the effects of a finite population mean that a more gradual evolution of fitness occurs, with correspondingly higher mutation rates.

These results suggests that there is a limit to the rate of environmental change which the self adaptive algorithm can respond to , this limit being related to the time needed for this phase transition to occur. This time will depend on the selection pressure, but also on the Innovation Rate since this determines both the background proportions of (initially) sub-optimal high mutation rates in the population prior to the change, and also the rate at which lower mutation rates are re-introduced into the population.

Finally it can be seen that Figure 5 explains another feature of the predicted behaviour of Figure 4, namely that the mean fitness of the population surpasses that of the stable eigenvector, before dropping to that level. This happens because during the second phase individuals with the lowest attached mutation rate dominate, and so the selection pressure is able to keep the population at a state with fewer "errors". This effect persists until the population is largely converged, and the other mutation rates are re-introduced by mutation, moving the population towards the stable state.

Conclusions

In this paper a model of a particular form of self-adaptation of mutation rates has been presented, along with empirical studies to investigate its applicability with finite populations. In this model, the mutation rate attached to an individual solution comes from one of a finite set of values, and at every time step can be changed to a randomly selected member of the set with

a fixed probability z. Although simpler than many of the models used by previous authors, this form of Self-Adaptation is shown to demonstrate the ability to adapt in both static and dynamic environments.

The empirical results presented show that the predicted fitness of the evolving population is overestimated, and reasons are given for this in terms of the discretisation of real populations with a granularity of $1/popsize$, and the greater number of equivalence classes in this model compared to that for a GA with a fixed mutation rate. Note that this presents another argument in favour of the particular form of self-adaptation used since it introduces a far fewer number of classes into the system than binary or Gray coded rates. However it is shown that the model does have predictive value, since the fitness levels at which evolutionary "epochs" occur correspond to the finite population spending time near eigenvectors of the system which lie close to the simplex and have a range of mutation rates present. Further work remainsto be done on predicting exactly which eigenstates will act as attractors.

When the behaviour of the system was studied after an environmental change, it was found that for a period of time there is a strikingly close match between the predicted and observed mean fitnesses over a range of population sizes. This has immediate benefits since it provides us with a means of predicting the ability of "real" algorithms to react to dynamic environments, and of tuning the range of mutation rates available, and the meta-mutation rate z so as to achieve desirable performance. This work is ongoing.

The problem studied here, the simplest example of a "function of unitation", was taken to illustrate a general approach. It would be perfectly possible to apply the techniques here to other such functions (see [5] for further examples of problems of this type), by making appropriate changes to the selection matrix. In [8] an example is given of how appropriate mixing matrices can be constructed for problems where blocks of genes ned to be considered together. This will be done in future work.

It would be possible to extend this approach to consider any problem type by considering the proportions of the population with particular genotypes, rather than using the aggregating approach. However there are two impediments. Firstly the size of the matrices induced has rendered this approach impractical for standard GAs, so with the number of states increased by a factor q, the size of problems that could be manipulated would be very small. Secondly, as has been seen above, the predictive power of the models depends on the closeness of the match between the actual samples of the problem space (i.e. successive finite sized populations) and the model. In practice this meant that population sizes of the order of the number of states were needed before the empirical results matched the predictions. This suggests that some form of aggregating approach, such as the one used in this paper, is required if the model is to have predictive value.

References

[1] T. Back. Self adaptation in genetic algorithms. In F. Varela and P. Bourgine, editors, *Towards a Practice on Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 263–271. MIT Press, 1992.

[2] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21. Lawrence Erlbaum, 1987.

[3] R. Hinterding, Z. Michalewicz, and T. Peachey. Self adaptive genetic algorithm for numeric functions. In H. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the Fourth Conference on Parallel Problem Solving from Nature*, pages 420–429. Springer Verlag, 1996.

[4] N. Krasnogor and J. Smith. Emergence of profitiable search strategies based on a simple inheritance mechanism. In *Proceedings of GECCO2001*. Morgan Kaufmann, 2001.

[5] J. Rowe. Population fixed-points for functions of unitation. In W. Banzhaf and C. Reeves, editors, *Foundations of Genetic Algorithms 5*, pages 69–84. Morgan Kaufmann, 1999.

[6] J. Smith and T. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the Third IEEE International Conference on Evolutionary Computing*, pages 318–323. IEEE Press, 1996.

[7] D. Stewart and Z. Leyk. *Meschach Library*. Australian National University.

[8] E. van Nimwegen, J. P. Crutchfield, and M. Mitchell. Statistical dynamics of the royal road genetic algorithm. *Theoretical Computer Science*, 229:41–102, 1998.

[9] M. D. Vose. *The Simple Genetic Algorithm*. MIT Press, 1999.

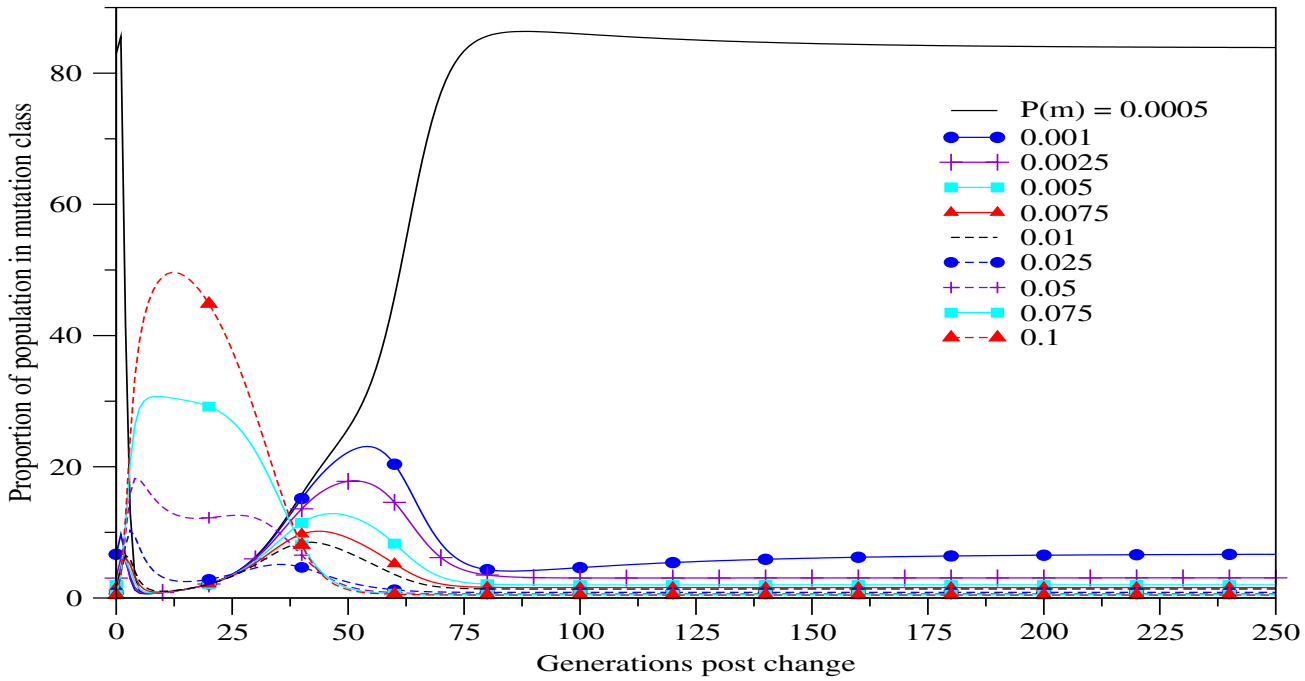


Figure 5: Predicted Evolution of Mutation Classes After Environmental Change

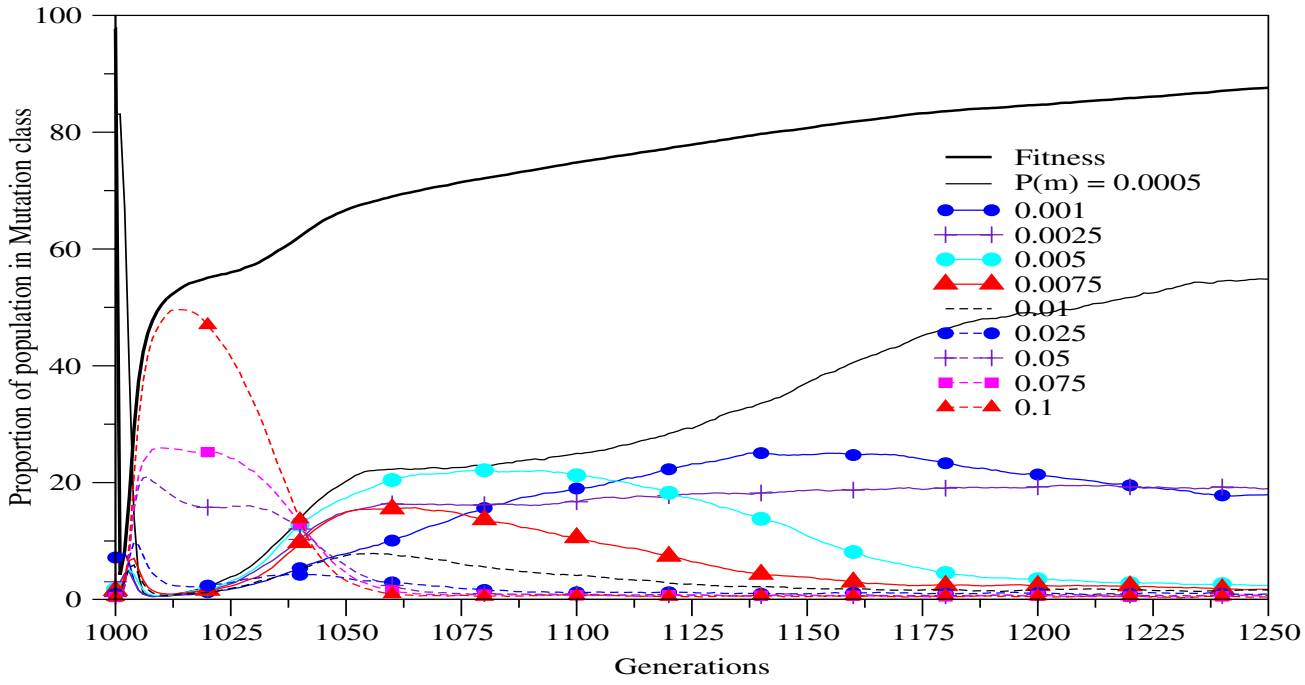


Figure 6: Observed Evolution of Mean Fitness and Mutation Classes After Environmental Change, Population 1000

A Generational Scheme for Partitioning Graphs

Alan Soper
Computing & Mathematical Sciences,
University of Greenwich,
Old Royal Naval College, Greenwich,
London, SE10 9LS, UK.
A.J.Soper@gre.ac.uk

Chris Walshaw
Computing & Mathematical Sciences,
University of Greenwich,
Old Royal Naval College, Greenwich,
London, SE10 9LS, UK.
C.Walshaw@gre.ac.uk

Abstract

Graph partitioning divides a graph into several pieces by cutting edges. Very effective heuristic partitioning algorithms have been developed which run in real-time, but it is unknown how good the partitions are since the problem is, in general, NP-complete. This paper reports an evolutionary search algorithm for finding benchmark partitions. Distinctive features are the transmission and modification of whole subdomains (the partitioned units) that act as genes, and the use of a multilevel heuristic algorithm to effect the crossover and mutations. Its effectiveness is demonstrated by improvements on previously established benchmarks.

1 INTRODUCTION

The graph partitioning problem can be stated as: partition the vertices of a graph into a given number of sets so that each set is of (approximately) equal size and so that the number of edges cut by the partition is minimised. The need for graph partitioning arises naturally in many applications such as distributing a finite element mesh across the nodes of a parallel computer in order to minimise communication overhead. It is well known that this problem is NP-complete (i.e. it is unlikely that an optimal solution can be found in polynomial time), so in recent years much attention has been focused on developing suitable heuristics, and a range of powerful methods have been devised, e.g. [8].

Here we report on a technique, combining an evolutionary search algorithm together with a multilevel graph partitioner, which has enabled us to find partitions considerably better than those that can be found by any of the public domain graph partitioning packages such as JOSTLE, METIS, etc. We do not claim this evolutionary technique as a possible substitute for the aforementioned packages; the very long run times preclude such a possibility for the typical applications in which they are used. However we do consider it of interest to find the best possible partitions for benchmarking purposes and for certain applications such as circuit partitioning, where the quality of the partition is paramount, the computational resources required may be completely justified by the very high quality partitions that the technique is able to find.

The main focus of this paper is to describe a strategy for combining evolutionary search techniques with a standard graph partitioning method. In Section 2 we outline the multilevel graph partitioning method used and establish notation & definitions. In Section 3 we then describe the genetic framework by defining the crossover and mutation operators and discuss how they are combined with the multilevel partitioner. Related work is also discussed here. We have conducted many experiments to test the technique and in Section 4 present some of the results including tests on unstructured meshes (§4.1). We also compare our results against a recent benchmark of Kang & Moon, [10]. Some of these graphs have similar structure to meshes, but some less structured examples are included.

The principal innovation described in this paper is the construction of crossover and mutation operators with an heuristic bias suitable for partitioning certain types of graphs which include meshes. These operators rely on the use of a multilevel graph partitioner, which is used to partition carefully chosen subgraphs of the original graph.

2 MULTILEVEL GRAPH PARTITIONING

Let $G = G(V, E)$ be an undirected graph of vertices V, with edges E. Given that the graph needs to be distributed to P processors, define a partition π to be a mapping of V into P disjoint subdomains S_p such that $\bigcup_P S_p = V$. The partition π induces a *subdomain graph* on G which we shall

refer to as $G_\pi = G_\pi(S, L)$; there is an edge or *link* (S_p, S_q) in L if there are vertices $v_1, v_2 \in V$ with $(v_1, v_2) \in E$ and $v_1 \in S_p$ and $v_2 \in S_q$. We denote the set of inter-subdomain or cut edges (i.e. edges cut by the partition) by E_c. Vertices which have an edge in E_c (i.e. those which are adjacent to vertices in another subdomain) are referred to as *border* vertices. Finally, note that we use the words subdomain and processor more or less interchangeably: the mesh is partitioned into P subdomains; each subdomain S_p is assigned to a processor p and each processor p owns a subdomain S_p.

In the context of partitioning a mesh for a parallel application, the definition of the graph partitioning problem is to find a partition which evenly balances the load or vertex weight in each subdomain whilst minimising the communications cost. To evenly balance the load, the optimal subdomain weight is given by $\overline{S} := \lceil |V|/P \rceil$[1] and the *imbalance* is then defined as the maximum subdomain weight divided by the optimal (since the computational speed of the underlying application is determined by the most heavily weighted processor). There is some discussion about the most appropriate metric for partitioning, e.g. [7], and indeed it is unlikely that any one metric is appropriate, however, it is common practice in graph partitioning to approximate the communications cost by $|E_c|$, the weight of cut edges or *cut-weight*. The usual (although not universal) definition of the graph partitioning problem is therefore to find π such that $|S_p| \leq \overline{S}$ and such that $|E_c|$ is (approximately) minimised.

In fact it has been noted for some time that partition quality can often be improved if a certain amount of imbalance is allowed, [15]. If we allow $\theta\%$ imbalance then the partitioning problem becomes 'find a partition π such that $|S_p| \leq \overline{S} \times (100 + \theta)/100$ and that $|E_c|$ is (approximately) minimised'.

2.1 The multilevel paradigm

In recent years it has been recognised that an effective way of both speeding up graph partitioning techniques and/or, perhaps more importantly, giving them a global perspective is to use multilevel techniques. The idea is to match pairs of vertices to form *clusters*, use the clusters to define a new graph and recursively iterate this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned (possibly with a crude algorithm) and the partition is successively optimised on all the graphs starting with the coarsest and ending with the original. This sequence of contraction followed by repeated expansion/optimisation loops is known as the multilevel paradigm and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan-Lin (KL), [12], and other optimisation algorithms. The multilevel idea was first proposed by Barnard & Simon, [2], as a method of speeding up spectral bisection and improved by both Hendrickson & Leland, [8] and Bui & Jones, [4], who generalised it to encompass local refinement algorithms. Several algorithms for carrying out the matching of vertices have been devised by Karypis & Kumar, [11], while Walshaw & Cross describe a method for utilising imbalance in the coarsest graphs to enhance the final partition quality, [18].

[1] where the ceiling function $\lceil x \rceil$ returns the smallest integer greater than x

3 THE GENETIC ALGORITHM

Genetic algorithms produce new search points by one of two operations: crossover which combines information from two or more randomly selected individuals in the current generation, and mutation which modifies a single, randomly selected, individual. The construction of successful crossover and mutation operators is problem specific and often complex, especially where individuals are subject to constraints (as are the partitions) so that information from different individuals cannot be arbitrarily combined or modified. Further, the information needs to be effectively exploited so that new individuals result that are fitter than the current best individuals with sufficient probability even when the current generation is already very good, [1].

A number of genetic algorithms for graph partitioning (e.g. [10]) have been constructed using a 'linear' chromosomal representation consisting of a list of subdomain memberships of a graph's vertices, each list item representing the subdomain in which the vertex appears. Crossover combines information from two chromosomes using standard operations (one-point crossover etc) to produce a child chromosome. In this case the linkage is determined by distance apart in the list and given that the ordering of vertices is arbitrary for most graphs, so is the linkage. The linkage has been improved by defining orderings of the list items which place nearby vertices in the graph (separated by few edges) close together in the list, and by 'normalising' the chromosomes before mating by relabeling the subdomains in one parent so that it has more vertices with the same subdomain membership as when they appear in the second, [10].

Genetic algorithms using this representation usually apply a local optimisation procedure to the resulting offspring, which improves and repairs them so that they are again balanced partitions. A novel and more powerful such procedure, termed Cyclic Partitioning, has recently been used by Kang & Moon with a GA of this type. Their procedure provides a more comprehensive search for local improvements than previous Kernighan-Lin based schemes by investigating the possible improvements available by transferring

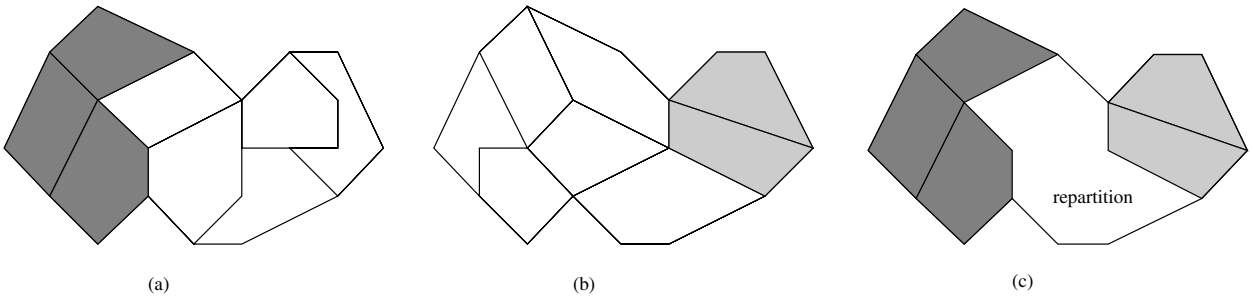


Figure 1: An illustration of the crossover operator

vertices across a set of partition boundaries (one vertex at each), such that the subdomains the vertices belong to form a cycle in the subdomain graph. They have run extensive tests comparing their genetic algorithm with recursive Kernighan Lin, pairwise Kernighan Lin and Cyclic Partitioning, using many repeated applications of the optimisation algorithm on different, initial, randomly-generated partitions. The authors have used the results to establish a set of high quality, benchmark partitions documented in [5]; 8-way & 32-way partitioning were performed.

Soper *et al.* have recently constructed a genetic algorithm which uses neither a linear chromosomal representation nor a traditional crossover operator, [16]. The crossover is implemented by modifying the graph to record where the parents had cut-edges by weighting them, and then applying a local optimisation procedure JOSTLE to the new graph so that cut edges of the parents are more likely candidates to be cut again due to the weighting. The mutation operator has an heuristic bias which exploits the local translational invariance possessed by many graphs of interest. This work produced benchmark partitions for evaluating public domain packages, and especially on graphs representing unstructured meshes. The current work is based on similar operators but further exploits the properties of the graphs being partitioned. The major difference is that the local optimisation procedure used during crossover and mutation needs only to be applied to a fraction – almost always less than half – of the graph to be partitioned. Much more information is transferred into the offspring from the parent(s) and the optimisation algorithm is more effectively focussed on one part of the problem at a time.

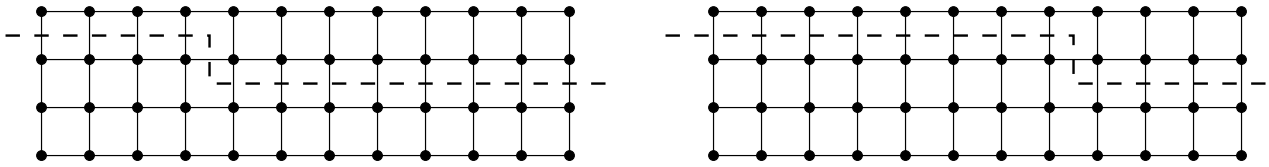
3.1 Recombining and mutating subdomains

Both crossover and mutations act on subdomains (or the set of of cut edges containing a subdomain). Crossover selects sets of complete subdomains from two individuals, and combines them in the child by partitioning the remainder of the graph as illustrated in Figure 1 ; Figures 1(a) & 1(b) show two parent partitions which have been selected for crossover. Sets of adjacent subdomains which do not intersect are selected (shown shaded) and the remainder of the graph – the unshaded part of Figure 1(c) – is repartitioned. Crossover seeks to exploit locality - the fact that graphs needing to be partitioned often only have vertices with low degree, showing local connectivity. This property holds for unstructured meshes which in their spatial embedding of physical origin only have short range connections, reflecting the locality of the physical systems they model. Locality allows subdomains from one individual to be successfully recombined with those from another when they are well separated.

Mutation takes a set of subdomains from an individual that constitute a cycle in the subdomain graph. The subgraph defined by this cycle is then repartitioned so as to exploit local translational symmetry; new partition boundaries are sought close to existing boundaries where they should have similar and so sometimes less cut edges. Another desirable property of mutations is that they are compatible or commute [14], i.e. their result does not depend on the order of their application. Our mutations will tend to have this property, either because their defining cycles don't intersect, or when they do because local translational symmetry, provides sufficient variations of common, partition boundaries to accommodate the balance constraint with a very similar number of cut edges.

In summary crossovers are constructed by producing cuts in the subdomain graphs of two individuals and mutations by constructing cycles in the subdomain graph of one individual. Figure 2 shows a case where a partially translated boundary has exactly the same number of cut edges.

Selection of subdomains for crossover: The number of subdomains selected from the first parent was chosen randomly and uniformly from the range $(P/4) - 1$ to $(P/2) - 1$, which choice prevented a parent from producing an offspring mostly identical to itself. The first subdomain was chosen randomly, then the second from its neighbours, the third from neighbours of both these subdomains, with probability proportional to the number of chosen neighbours (1 or 2), and so on. Thus there is a bias to choosing sets of subdomains with more internal or common partition bound-



(a) (b)

Figure 2: A translated boundary fragment with the same number of cut edges

aries. The choice of a more compact structure increases the chances of successful recombination; the extreme opposite a collection of scattered non-neighbouring subdomains would effectively prevent any substantial change in the parent.

Subdomains are selected from the second parent by a process of elimination. First delete subdomains from this parent which have any vertices in common with the subdomains selected from the first. Then delete those that have more than two fifths of their vertices in common with the neighbours of the subdomains selected from the first parent. The remaining subdomains are included in the offspring. The heuristic provides a balance between the competing demands of information transfer, i.e. copy more subdomains and their bordering cut edges into the offspring, and the need to allow successful recombination, i.e. is the remainder of the graph capable of being partitioned with a small enough number of cut edges?

In general, the crossover offspring partition will not be of sufficiently high quality to be accepted into the succeeding generation of the genetic algorithm, therefore it is immediately subject to a hill-climbing sequence of mutations.

In some cases during crossover, it is possible for the subdomains selected from the two parents to be such that the remaining subgraph cannot be partitioned within the imbalance constraint since it contains too many vertices. Such situations turn out to be rare however and the crossover is abandoned.

Selection of cycles of subdomains for mutations: Rather than selecting a cycle independently for each mutation, sets of mutations are carried out together in a hill-climbing sequence, the result of a mutation being the starting point of the next if it produces a better or equally good partition with respect to the number of cut edges. If the partition is worse, the mutation is ignored.

A random spanning tree of the subdomain graph is generated, and then the fundamental cycles with respect to this are recorded. Of these, cycles with lengths less than 4 and greater than 8 are discarded; small cycles because they allow little variation and larger cycles since it is more difficult for the partitioner to simultaneously improve more boundaries. When a cycle of subdomains is selected borders between subdomains are also targets for improvement, so that very small cycles tend to be included in the optimisation process already. Variations over longer cycles are provided by the joint effect of crossover and mutation - they will tend to be cut on crossover, and the resulting parts improved as part of other smaller cycles.

Thus a hill-climbing sequence is the set of mutations associated with the remaining fundamental cycles. These sequences are used since they are more efficient to implement than producing the mutations individually and their cycles will include most subdomain boundaries.

3.2 Partitioning Subgraphs

The implementation of the new partitions of subgraphs needed for both crossover and mutation are based on previous work, [16]. We use a multilevel technique as an efficient and effective partitioner. In fact the multilevel partitioner used is known as JOSTLE and we shall henceforth refer to it as such, although any graph partitioning heuristic which can deal with real (non-integer) edge weights could be used. JOSTLE is fully described in [18].

Both crossover and mutation require that some edges of the graph be made more likely to appear as cut edges under the action of JOSTLE. This is achieved by biasing the costs of the edges: the cost of an edge becomes unity plus a positive number and JOSTLE takes account of these additional costs when seeking low cost partitions. Mutations are implemented by making existing cut edges and their neighbours much less costly and crossover by making the cut edges of both parents occurring in the subgraph being repartitioned slightly less costly. New biases are explicitly and partially randomly constructed from the parent(s) for each operation.

3.3 The CHC adaptive search algorithm

The genetic algorithm framework chosen was Eshelman's CHC adaptive search algorithm, [6]. It has been shown to work successfully on a wide range of problems (e.g.

[13, 17]) with the same parameter settings and, importantly for partitioning large graphs, it uses a small population of 50 individuals. This allowed the simulations to run in a computer's memory. Its main features are: an elitist selection strategy, a highly explorative crossover operator, incest prevention and partial randomisations or restarts.

We adapted CHC as follows: Since the crossover provides less variation than that used in the original version of CHC, we also allow mutations. When a pair of parents are selected for mating and pass the incest test, they crossover with probability 0.3 and suffer mutation the rest of the time. When mutation only is applied, a separate offspring is not produced, rather, provided an improved partition or one of equal quality results (compared to the parent), it is overwritten. This procedure helps maintain diversity within the population.

When preventing incest, the distance between any two individual partitions was defined to be the number of vertices in the graph minus the number of edge vertices that they have in common. This measure clearly takes common cut edges into account, but also any nearby borders. The distance threshold is initialised, both when starting the genetic algorithm and on restarts, to the average distance apart of some randomly sampled pairs in the population. Clearly the distance between individuals is never zero, so that a distance threshold to initiate restarts has to be set. This is taken to be the distance of the best individual from itself, the expected distance apart of individuals in the population when it has converged. The distance threshold was decremented by 10 whenever no new offspring were accepted. This number need not be tuned to any great accuracy, since a small value will produce earlier subsequent decrements and vice-versa. However the value should be large enough to allow more parents to mate on average; a decrement of 10 allowed this.

At a restart the best individual is randomised by mutating the whole partition as described above, but with a heavier bias supplied to non-border vertices in order to retain approximately 65% of the border vertices. Three restarts are allowed after which the genetic algorithm is reinitialised.

The fitness of an individual was defined to be minus the product of the number of cut edges times the imbalance. JOSTLE occasionally produces partitions violating the balance constraint which are strongly penalised by this scheme.

The initial population was produced by repeatedly partitioning the graph with JOSTLE using random but small biases, of the order of 0.1.

4 EXPERIMENTAL RESULTS AND DISCUSSION

We have implemented the algorithms described here within the framework of JOSTLE, a mesh partitioning software tool developed at the University of Greenwich and freely available for academic and research purposes under a licensing agreement[2]. The experiments were carried out on a variety of different machines; with its very long runtimes (of several days in the case of the larger graphs), the evolutionary search approach can soak up CPU cycles and the tests were run so as to use up any spare capacity in the system. As a result we have not measured runtimes.

4.1 Results on unstructured meshes

Table 1: A summary of the test graphs

	size		degree			
graph	V	E	$\leq$	$\geq$	avg	type
data	2851	15093	17	3	10.6	3D nodal
3elt	4720	13722	9	3	5.8	2D nodal
uk	4824	6837	3	1	2.8	2D dual
ukerbe1	5981	7852	8	2	2.6	2D nodal
add32	4960	9462	31	1	3.8	circuit
crack	10240	30380	9	3	5.9	2D nodal
4elt	15606	45878	10	3	5.9	2D nodal

The test graphs have been chosen to be a representative sample of small to medium scale real-life problems and include mostly 2D (and one small 3D) examples of nodal graphs (where the mesh nodes are partitioned) and dual graphs (where the mesh elements are partitioned). The test suite also includes one non mesh-based graph, add32.

Table 1 gives a list of the graphs, their sizes, the maximum, minimum & average degree of the vertices and a short description. The degree information (the degree of a vertex is the number of vertices adjacent to it) gives some idea of the character of the graphs. These range from the relatively homogeneous dual graphs, where every vertex represents a mesh element, in these cases a triangle and so every vertex has at most 3 or 4 neighbours respectively, to the non mesh-based graph such as add32 which has vertices of degree 31. As the graphs are not weighted, the number of vertices in V is the same as the total vertex weight $|V|$ and similarly for the edges E.

Graph partitioning algorithms can usually find higher quality partitions if the balancing constraint is relaxed slightly. Indeed some of the public domain graph partitioning packages such as JOSTLE & METIS have an in-built, although adjustable, imbalance tolerance of 3% (i.e. the largest subdomain is allowed to be up 1.03 times the size of the maxi-

[2] available from `http://www.gre.ac.uk/jostle`

mum allowed for perfect balance). We therefore tested the evolutionary algorithm with various tolerances and Table 2 shows a comparison of the cut-weight results with 0% and 3% imbalance tolerances, C_E^0 and C_E^3 respectively, for four values of P (the number of processors/subdomains). For each value of P, the first & second columns show the cut-weight with the allowed imbalance, while the third column shows the ratio of cut-weight for 3% imbalance scaled by that for 0% imbalance, C_E^3/C_E^0. Thus the figure of 0.98 for the data graph and $P = 8$ means that the algorithm was able to find a partition 2% better if allowed a 3% imbalance tolerance. As can be seen, the improvement in quality for these tests is up to 7% and on average is around 3%.

To demonstrate the quality of the partitions, we have compared the results in Table 2 with those produced by a public domain partitioning package JOSTLE (JOSTLE 2.2, March 2000), [18]. Firstly Table 3 shows a comparison of the cut-weight results for the public domain version of JOSTLE compared to the evolutionary search algorithm. These results are an improvement on our previous evolutionary search implementation, [16]. The average difference in the quality ranges from 23% to 20% as P increases and can be as bad as 75%. Note that differences in quality tend to diminish as P increases. It is tempting to speculate that this is because the margins for difference decrease as the number of vertices per subdomain ($\approx V/P$) decreases. Indeed in the limit where $V = P$ the only balanced partition (for an unweighted graph at least) is to put one vertex in each subdomain and so the differences vanish altogether.

4.2 Comparison with the results of Kang & Moon

In this section we compare our results against a recent benchmark of Kang & Moon, [10]. Some of these graphs have similar structure to meshes, but some less structured examples are included, [5, 9].

Three types of graph were tested: U$n.d$, random geometric graphs of n vertices that lie in the unit square and whose co-ordinates are chosen uniformly from the unit interval; Grid$n.b$, a grid graph of n vertices whose optimal bisection size is known to be b and W-grid$n.b$, the same graph with wrapped boundaries; Breg$n.b$, a random regular graph of n vertices each of which has degree 3, and the optimal bisection size is b with probability $1 - o(1)$, [3].

We expect the random geometric graphs and grid graphs to be suitable for the crossover and mutation operators because of their geometric origin – they both arise from the embedding of graph vertices in a low dimensional Euclidean space, with only local connections between points giving rise to edges. The randomly generated Breg$n.b$ graphs, with edges possible between any pairs of vertices do not exhibit the structure required by the heuristic bias of the genetic algorithm. Caterpillar graphs were not used since they have a very different structure altogether.

Kang & Moon's benchmarks were produced by running their genetic algorithm 50 times on each problem graph, with each run given an allotted CPU time and keeping the best. The objectives of our experiments were twofold. Firstly to test whether our genetic algorithm was robust – could it find partitions as good as those of Kang & Moon without requiring repeated runs (or equivalently the repeated full reinitialisations after 3 restarts) within broadly similar total time budgets. This is a good test of the heuristic bias given to the crossover and mutation operators, since if insufficient very fit offspring are produced, the population will converge and the quota of 3 restarts soon used up. Secondly to support, improve and extend their benchmarks. For 32-way partitioning the number of vertices in the graphs did not divide exactly by 32, so that some partitions will have more vertices than others. We use a less restrictive constraint than theirs, which requires that the partitions differ by no more than one vertex, since we only constrain the maximum allowed number of vertices, so for 32-way partitioning we are extending the benchmark. For 8-way partitioning the number of vertices divides exactly, so our constraint is the same and hence we can justly claim to support and improve on their results.

The same parameters were used on all experiments except that for the W-grid5000.100 and U1000.40 graphs, the ratio of crossover to mutation was increased from 3::7 to 7::3. This slowed the rate of convergence of the population, allowing a more thorough search and providing evidence for the effectiveness of the crossover operator. Table 4 shows our results, giving the minimum number of cut edges found (with the change relative to those of Kang & Moon in brackets) and the number of subgraph evaluations taken to find the result.

For 8-way partitioning the results support or improve on those of Kang & Moon, except for the graph Breg5000.16. Even though improved results were found for the remaining examples of the Breg graphs, they required substantially more subgraph evaluations to find results as good as those of Kang & Moon, in agreement with our expectations of performance on this type of graph.

For 32-way partitioning, because of our less exacting constraint on the imbalance, we expected to find less cuts than the previous benchmark. This turned out to be the case, except for the Breg graphs. More evaluations were allowed for these graphs, since the genetic algorithm converged very slowly, showing further potential. Breg5000.0, the most suitable for our algorithm given its construction, eventually yielded less cut edges than the more constrained partition of Kang & Moon. Our genetic algorithm made much slower progress towards partitions of similar quality

to their benchmark on the Breg graphs.

The robustness of our genetic algorithm was confirmed by its requiring only one run to match the benchmark in almost all cases.

5 SUMMARY & FUTURE WORK

We have described and tested an evolutionary search algorithm for partitioning graphs and reported new benchmark partitions that it found. Distinctive features are the transmission and modification of whole subdomains (the partitioned units) that act as genes, and the use of a multilevel heuristic algorithm to effect the crossover and mutations. These features implement an heuristic bias suitable for graphs such as unstructured CFD meshes and their effectiveness is demonstrated by improvements on previously established benchmarks.

In future we aim to look at the integration of the evolutionary search procedure more fully into the multilevel framework. We also intend to study more carefully the parameters which govern the interaction between the multilevel scheme and the evolutionary algorithm.

References

[1] L. Altenberg. The Schema Theorem and Price's Theorem. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 23–49. Morgan Kaufmann, San Mateo, 1995.

[2] S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.

[3] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph Bisection Algorithms with Good Average Case Behavior. *Combinatorica*, 7(2):841–855, 1987.

[4] T. N. Bui and C. Jones. A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. In R. F. Sincovec *et al.*, editor, *Parallel Processing for Scientific Computing*, pages 445–452. SIAM, Philadelphia, 1993.

[5] T. N. Bui and B. R. Moon. Genetic Algorithms and Graph Partitioning. *IEEE Trans. Comput.*, 45(7):841–855, 1996.

[6] L. J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, San Mateo, 1991.

[7] B. Hendrickson and T. G. Kolda. Graph Partitioning Models for Parallel Computing. *Parallel Comput.*, 26(12):1519–1534, 2000.

[8] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In S. Karin, editor, *Proc. Supercomputing '95, San Diego*. ACM Press, New York, NY 10036, 1995.

[9] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: Part I, Graph Partitioning. *Oper. Res.*, 37(6):865–892, 1989.

[10] S. Kang and B. R. Moon. A Hybrid Genetic Algorithm for Multiway Graph Partitioning. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 159–166. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[11] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[12] B. W. Kernighan and S. Lin. An Efficient Heuristic for Partitioning Graphs. *Bell Syst. Tech. J.*, 49:291–308, 1970.

[13] K. Mathias, L. Eshelman, J. D. Schaffer, L. Augusteijn, P. Hoogendijk, and R. van de Wiel. Code Compaction Using Genetic Algorithms. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 710–717. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[14] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag, Berlin, 1982.

[15] H. D. Simon and S.-H. Teng. How Good is Recursive Bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, 1997.

[16] A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Approach to Graph Partitioning. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 674–681. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[17] M. Vazquez and D. Whitley. A Hybrid Genetic Algorithm for the Quadratic Assignment Problem. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 135–142. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[18] C. Walshaw and M. Cross. Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. *SIAM J. Sci. Comput.*, 22(1):63–80, 2000. (originally published as Univ. Greenwich Tech. Rep. 98/IM/35).

Table 2: A comparison of cut-weight results for the evolutionary search algorithm with 0% and 3% imbalance tolerances, C_E^0 and C_E^3 respectively

	$P=8$			$P=16$			$P=32$			$P=64$		
graph	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$
data	671	656	0.98	1135	1118	0.99	1811	1783	0.98	2859	2795	0.98
3elt	348	336	0.97	596	565	0.95	963	949	0.99	1553	1524	0.98
uk	91	86	0.95	152	144	0.95	263	249	0.95	419	412	0.98
ukerbe1	113	111	0.98	201	196	0.98	338	334	0.99	548	542	0.99
add32	71	68	0.96	126	117	0.93	218	212	0.97	544	520	0.96
crack	683	683	1.00	1091	1076	0.99	1703	1662	0.98	2603	2519	0.97
4elt	552	529	0.96	946	909	0.96	1571	1530	0.97	2618	2552	0.97
Average			0.97			0.96			0.98			0.98

Table 3: A comparison of cut-weight results for JOSTLE, C_J^3, against those of the evolutionary search algorithm, C_E^3, both with 3% imbalance tolerance

	$P=8$		$P=16$		$P=32$		$P=64$	
graph	C_J^3	$\frac{C_J^3}{C_E^3}$	C_J^3	$\frac{C_J^3}{C_E^3}$	C_J^3	$\frac{C_J^3}{C_E^3}$	C_J^3	$\frac{C_J^3}{C_E^3}$
data	756	1.15	1263	1.13	2106	1.18	3140	1.12
3elt	418	1.24	603	1.07	1020	1.07	1666	1.09
uk	106	1.23	180	1.25	315	1.27	490	1.19
ukerbe1	121	1.09	233	1.19	378	1.13	593	1.09
add32	106	1.56	180	1.54	257	1.21	909	1.75
crack	751	1.10	1191	1.11	1804	1.09	2733	1.08
4elt	656	1.24	1012	1.11	1687	1.10	2772	1.09
Average		1.23		1.20		1.15		1.20

Table 4: The results of the evolutionary search algorithm with a 0% imbalance tolerance on the partitioning benchmark graphs showing the cut-weight, $|E_c|$, and the number of subgraph partitions required to find it

	$P=8$		$P=32$	
graph	$\|E_c\|$	# evals	$\|E_c\|$	# evals
Grid1000.20	114 (-0)	6932	302 (-12)	52183
Grid5000.100	250 (-0)	19639	658 (-1)	437063
W-grid1000.40	172 (-4)	4566	372 (-12)	246908
W-grid5000.100	400 (-0)	215090	811 (-9)	1342821
U1000.10	180 (-7)	57350	559 (-18)	463057
U1000.20	812 (-0)	10411	2325 (-42)	327932
U1000.40	2562 (-0)	70147	7241 (-88)	280541
Breg5000.0	1079 (-17)	457009	1675 (-45)	4991004
Breg5000.4	1081 (-12)	498954	1779 (+54)	2915867
Breg5000.8	1079 (-19)	450115	1786 (+49)	2814953
Breg5000.16	1180 (+103)	1148758	1755 (+62)	2941382

A Genetic Algorithm with Multiple Reading Frames

Terence Soule
Department of Computer Science, University of Idaho
Moscow, ID 83844-1010
email: tsoule@cs.uidaho.edu
phone: 208-885-7789

Amy E. Ball
Moscow, ID 83843
Email: aeball@att.net

Abstract

Multiple reading frames are an important feature of gene expression in biological systems. Multiple reading frames allow several genes to be encoded in the same region of DNA. This produces an inherent form of information compression. In some organisms this compression is so extensive their genes are effectively longer than their DNA. In this paper a modification of a simple genetic algorithm (GA) is introduced that uses multiple reading frames. It is shown that some information compression does occur. Interestingly, while the GA does utilize information compression where necessary there is evidently strong evolutionary pressure to limit this compression.

1 Introduction

Recently there has been considerable interest in applying the basic processes of biological gene expression to artificial forms of evolution, most notably genetic algorithms (GAs). Gene expression, the process by which DNA produces proteins, is a complex process which has a significant impact on how DNA actually 'produces' an organism. Several researchers have explored the impact of artificial gene expression on artificial evolution (see [1] for a summary of this work) leading to potentially beneficial modifications of the basic GA and other artificial evolutionary models.

One important feature of the gene expression process that has received limited attention from the evolutionary computation field is multiple reading frames. Multiple reading frames occur because there are three distinct 'translations' of a given segment of DNA. Each of these translations can encode different proteins. Thus, DNA is inherently capable of a certain amount of information compression.

In this paper we examine a GA that uses an analogy of gene expression that allows multiple reading frames. The primary questions are whether a GA can use reading frames and whether a GA can exploit the information compression capability inherent in multiple reading frames.

2 Background

The primary genetic material of most biological organisms is Deoxyribonucleic acid (DNA). DNA itself is composed of four nucleic acids: adenine (A), cytosine (C), guanine (G), and thymine (T). Every protein component of an organism is made from a long sequence of As Cs Gs and Ts.

Interestingly, most of the DNA contained in a typical chromosome does not contain instructions for making proteins or other functional products. Much of this non-coding DNA serves a structural purpose or serves to regulate how and when, the coding portion of the DNA is utilized or to facilitate the coding process itself. Finally, much of the non-coding DNA serves no known purpose.

The coding portion of DNA is organized into sets of three nucleic acids known as codons. Thus, there are 64 codons in the genetic code of all biological organisms (4^3). Each of the 20 amino acids is specified by at least one codon, which is interpreted in a complex two-step process. In the first step, transcription, the genetic code embedded in DNA is reinterpreted as messenger ribonucleic acids (mRNAs). In the second step, translation, the mRNA is read and used to construct protein. Because the coding portions of DNA are separated on a chromosome, the function of some of the codons is to initiate or to terminate the transcription process. The genetic code specifies 3 termi-

nation or "stop" codons and one initiation or "start" codon. The remaining 60 codons specify 19 of the 20 amino acids that make up proteins. The 20th amino acid, methionine, is specified by the start codon itself. (In fact, the initiation of transcription is dependent upon the presence of several specific but non-coding regions of DNA that must be adjacent to the start codon. Only when these DNA regions are present does the "start" codon actually initiate transcription, otherwise it specifies only the inclusion of methionine in the final protein.) There is considerable redundancy in the genetic code, with 61 codons specifying 20 amino acids. Only methionine is specified by a single codon. The remaining 19 amino acids are each specified by as few as two or as many as six codons. There are many useful sources for a further review of the biological genetic code and the processes of transcription and translation (see, for example the summary by Kargupta [2], or a genetics text such as *Genetics* [3]).

The arrangement of the genetic code into functional codons each consisting of three contiguous nucleic acids leads to a phenomenon known as reading frames. Because one can start reading the DNA sequence at either the first, second, or third nucleic acid of a codon there are three different possible sequences each of which can be interpreted separately. For example the DNA sequence:

...AACGTTGACTGCTAGTTTCACATGCGTACT...

can be organized into three distinct sets of codons,

... AAC GTT GAC TGC TAG TTT CAC ATG CGT ACT ...
...A ACG TTG ACT GCT AGT TTC ACA TGC GTA CT...
...AA CGT TGA CTG CTA GTT TCA CAT GCG TAC T...

(Any other reading frame will produce codons that are a subset of one of the primary three reading frames.) The fact that three full and unique coding sequences can be found in one DNA strand allows for a great deal of information compression, as several genes can overlap within the strand of DNA by using different reading frames.

In general, it is unclear to what extent most biological organisms take advantage of multiple reading frames to compress the genetic information contained in their DNA. However, there is precedence for such a process occurring in viruses, bacteria, and eukaryotes. Notably, the bacteriophage ϕX174 includes many regions where two and three genes overlap. The overlapping is so extensive that the total coding length of the bacteriophage is longer than its DNA[4].

The floating building block representation introduced by Wu and Lindsay shares many features with the biological process outlined above [5]. The representation uses a start tag to mark the beginning of genes and an identity tag that determines the 'function' of the gene. The length of the genes are fixed so no end tag (stop codon) is required.

In the floating building block representation a second start tag may appear within an 'active' gene. This leads to overlap similar to that produced by multiple reading frames. Additionally, the use of identity tags allowed some genes to be over or under specified.

Experimental results with the floating representation were positive, particularly for longer chromosomes. It performed significantly better than a standard GA on the two test functions, Royal Road and a version of symbolic regression. Much of the benefits seemed to arise because longer chromosomes allow additional over specification of genes and hence additional exploration of the search space within the same size population. In addition, multiple floating genes gave the GA more flexibility to explore different physical arrangements of the genes. Although the floating representation allows genes to overlap, the experiments did not examine the extent of overlap or how it changed over time.

Messy GAs also bear some resemblance to biological genes (see for example [6]). In messy GAs each 'gene' has an associated tag that determines the gene's location in a transcribed version of the genome. This allows the genes to be rearranged on the evolving chromosome.

3 The Encoding

Multiple reading frames require an encoding scheme that produces codons. We have attempted to write the simplest possible encoding rules that capture the essential properties of biological reading frames. In particular, the encoding is designed to include variable length genes using start and stop codons, potentially overlapping genes, and non-coding regions.

As noted above, nature uses 64 (4^3) codons. Our chromosomes are binary strings, so we choose a codon length of 5 which also produces 64 (2^5) codons. Of course, there is nothing obviously significant about the value 64. It was chosen simply to be similar the biological value. Non-binary encodings and other codon lengths could certainly be used and may be beneficial for some problems.

Codons were transcribed as follows:

$$11111 = \text{start}$$
$$00000 = \text{stop}$$
$$\text{ijklm} = \text{jklm for all other cases}$$

The start and stop sequences (11111 and 00000) act as regulatory codons. The other codons create a four bit sequence by dropping the first bit of the five bit codon. This creates two encodings for each 4 bit sequence (jklm can be encoded as 0jklm or 1jklm) except 0000 and 1111, which have one code apiece (10000 and 01111 respectively). The region between a start codon and a stop codon is equivalent to a biological gene. Although very simple, this encoding captures the essential properties of the genetic code necessary to study multiple reading frames.

Because our codons are five bits long there are five separate reading frames. The genes are evenly distributed among the reading frames. Thus, for N genes reading frame 1 specifies genes 1 through N/5, reading frame 2 specifies genes N/5+1 through 2N/5, etc. For example, our GA is tested on several function optimization problems (see Section 4). Each function depends on 25 variables. Thus, 25 values need to be specified. Each value is specified by a single gene, so there are 5 genes per reading frame.

Transcription of the first gene begins at the first start codon and continues until a stop codon or the end of the chromosome is reached. Transcription of the next gene begins at the next start codon and so forth, until the end of the chromosome is reached or all genes of that reading frame have been specified. If the end of the chromosome is reached before all of the gene are specified, the remaining variables are unspecified. Regions of the chromosome which do not fall between start-stop pairs are non-coding. Start codons in the middle of an open reading frame are ignored and thus are also non-coding.

In our experiment, each of the transcribed strings represents the value of one of the variables written in Gray code. Because the start and stop codons can occur at arbitrary locations the binary strings produced by the transcription process have arbitrary lengths. To avoid overflow errors at most the first 64 bits of the transcribed string are considered. Because the transcribed strings may have different lengths their range of possible values is variable. So, each transcribed string is scaled to a value between zero and one (by dividing by $2^{the\ string's\ length}$) and then is rescaled to the range appropriate to the problem.

Our encoding varies from other gene-like encodings in several significant ways. It combines floating genes with variable length genes. This gives the GA more control over the amount of genetic overlap (and hence the potential amount of information compression) than other encodings. It does not use identity tags, so over specification is not a possibility (although under specification is, if there are too few start codons). Our experiments focus on how the potential problems and advantages of overlapping genes are handled by the evolutionary algorithm.

4 Test Problems

We tested our GA on modifications of four of the functions from the De Jong test suite. We used functions F1, F3, F7 (Schwefel's function), and F8 (Griewangk's function) [7, 8]. The goal is to find the function maximums. The fitness of an individual is the returned value.

Our modified functions were:

$$f_1(x_i) = \sum_{i=1}^{25} x_i^2 \qquad x_i \epsilon [-5.12, 5.12] \tag{1}$$

$$f_3(x_i) = \sum_{i=1}^{25} integer(x_i) \qquad x_i \epsilon [-5.12, 5.12] \tag{2}$$

$$f_7(x_i) = \sum_{i=1}^{25} x_i sin(\sqrt{|x_i|}) \qquad x_i \epsilon [-512, 512] \tag{3}$$

$$f_8(x_i) = \sum_{i=1}^{25} x_i^2/4000 - \prod_{i=1}^{25} cos(x_i/\sqrt{i}) + 1 x_i \epsilon [-512, 512] \tag{4}$$

Clearly reading frames are only meaningful when there are sufficient genes to make use of the reading frames. Thus, each of the functions has been modified from the original version to include 25 variables. The value 25 was chosen strictly for convenience and consistency.

5 The Genetic Algorithms

These experiments compare a standard GA to a GA using codons and multiple reading frames. The parameters common to both GAs are summarized in Table 1. Both versions of the GA are generational.

For test functions F1, F2 and F3 the chromosome length is 250 bits. For the standard GA this meant each gene (variable value) is represented by exactly

Population Size	200
Crossover rate	0.8
Crossover type	one point
Mutation rate	1/Chromosome length
Selection	3 member tournament
Type	Generational
Elitism	1 member
Trials	30

Table 1: Summary of the parameters used with the GA with and without reading frames.

10 bits. For the GA with reading frames there are roughly 1250 bits available, or 50 bits per gene. (5 reading frames with 250 bits apiece. The actual number is slightly smaller because the reading frames lose a few bits at the beginning and end of the chromosome.)

Of course, with reading frames the number of bits encoding each solution is variable and in practice is much smaller than 50. Much of the chromosome may be noncoding. Some of the bits must be start and stop codons and the transcription process causes a 'loss' of 1/5 of the bits. In fact, the results show that the number of bits per variable evolves in a systematic way over the course of a GA run.

For F8 the chromosome length was 400. The standard GA uses 16 bits per variable. The GA with reading frames was subject to the same considerations discussed above.

6 Results

We begin by examining the performance of the GA on our test problems. Although our primary interest is how evolution arranges the genes in different reading frames, it is important that the GA perform reasonably. Additionally, the evolution of fitness does provide some insight into how the GA is adapting the reading frames.

The results of the GA and the GA with reading frames, averaged over 30 trials, are shown in Table 2. The GA using reading frames produces significantly better solutions than the standard GA in two of the four test cases (F1 and F8) and better average solutions in one case (F8). These data show that a GA can successfully use reading frames despite the added complexity of reading frames.

Additionally, there seems to be a specific reason for the poor performance on F3. For F3 the optimal solution is a maximum value (5.12) for all variables. This corresponds to a string of all ones, e.g. 11111111. When reading frames are used the sequence 11111111 is encoded as 01111 01111. However, this only produces an optimal solution in the first reading frame. In the second reading frame the sequence 01111 01111 is read as 0 11110 1111 and in the third frame it is read as 01 11101 111. Thus, for this problem the optimal solution for one reading frame is non-optimal for the other frames.

As a test we used a the following modification of the F3 function:

$$f_3'(x_i) = \sum_{i=1}^{25} |integer(x_i)| \qquad (5)$$

This function has several more potential solutions than the standard F3 function. Thus, it should be easier for the GA to find overlapping sequences within optimal (or near optimal) genes. The results are also shown in Table 2 and the GA with reading frames does outperform the standard GA. These results suggest that for some problems, such as F3, the encoding of the optimal solution(s) may make it extremely difficult to find optimal genes with overlapping regions.

Figure 1 shows the fitness of the best individuals (averaged over 30 trials) evolved with the standard GA and with reading frames on function F8. Initially the GA with reading frames performs much more poorly. Under specification of the variables is a likely cause of this poor initial performance.

Thus, we examined exactly how many variables are being specified by the GA with reading frames. This corresponds to the number of genes (or start-stop pairs) the GA is producing. Clearly, whether the GA can evolve sufficient genes across multiple reading frames is an important question. Figure 2 shows the number of variables specified at each generation for F1.

Initially, the average individual has approximately five of the twenty-five variables specified. This low value occurs simply because in randomly generated binary strings of 250 bits there will be relatively few start-stop pairs. However, over the course of evolution the number of specified variables grows steadily, almost reaching 24.97 for the best individuals and 24.24 for the average individual. Additionally, the best individuals consistently have more variables specified than the average. This is a clear indication that there is strong pressure on the GA to specify more of the variables.

The results for the other three test problems are similar and are summarized in Table 3. For the F8 function the initial number of specified variables is higher (14.3 for the best individual and 7.44 for the average individ-

	Best individual			Average individual		
	Standard GA	Reading frames	p	Standard GA	Reading frames	p
F1	625	642	< 0.01	613	607	< 0.1
F3	112	67.7	< 0.01	108	61.3	< 0.01
F7	19600	16900	< 0.01	19300	16400	< 0.01
F8	1550	1600	< 0.01	1600	1570	< 0.01
F3'	112	121	< 0.01	109	114	< 0.1

Table 2: Results at generation 50 for the best and average individuals. The raw numbers represent the optimal value achieved and are averaged over 30 trials. The p value is generated using Student's two-tailed test.

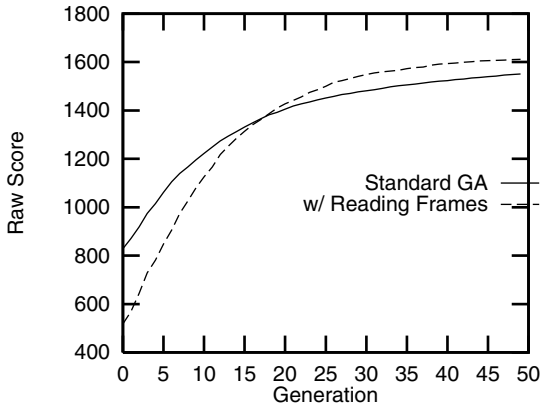


Figure 1: Fitness of the best individuals evolved with and without reading frames. These data are the average of 30 trials. The poor initial performance of the GA with reading frames is attributed to its failure to specific all of the variables.

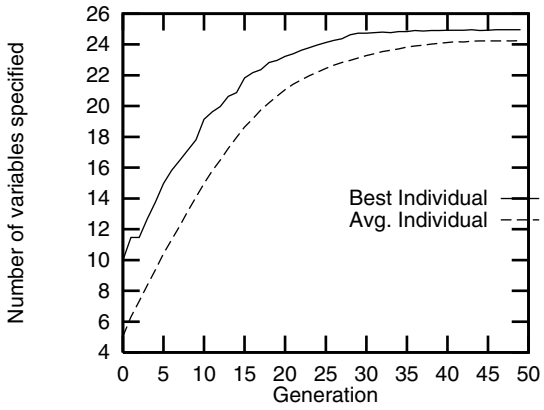


Figure 2: Number of variables specified, or number of genes per chromosome, for problem F1. Initially many of the variables are under specified due to a shortage of start and stop codons, but as evolutions proceeds the GA succeeds in specifying (almost) all 25 variables.

ual) simply because the chromosome is longer, which allows more start-stop pairs in the initial random chromosomes. Thus, a second feature of the GA with reading frames is that it can introduce new genes as necessary, by increasing the number of regulatory codons in the chromosomes. Similar behavior has been seen with messy GAs and floating building blocks, but is more difficult here because of the requirements to produce both start and stop codons and the complications introduced by overlap between the reading frames.

Finally, we consider the issue of information compression. Given that the GA is introducing additional genes, how long are these genes and how much do they overlap? To explore this question we focus on functions F7 and F8. These are the most interesting functions in that they are reasonably complex and they lead to dramatically different performance by the GA with reading frames.

Figure 3 shows the average length of the genes, when all genes are specified, for problems F7 and F8. (This is the length in bits, not including the start and stop codons.) In the early generations none of the solutions are fully specified and no data is shown. As noted above, the chromosomes used with F8 are longer and therefore include more start-stop pairs. Thus, fully specified solutions are found sooner.

For both problems the average gene length decreases over time. However, it is clear that this decrease is much faster and longer lasting for F8. (These runs were extended to 100 generations to illustrate this difference.) By the final generation the length of the genes for F8 has declined below 5, implying that some

Test Problem	Average number of variables specified initially	Average number of variables specified in generation 50
F1	5.04	24.2
F3	5.05	22.9
F3'	5.09	24.3
F7	5.09	24.3
F8	7.44	24.7

Table 3: Average number of specified variables in the initial and final generations. By the final generation the GA has introduced sufficient regulatory codons to specify almost all of the variables.

of the genes consist of only a start-stop pair.

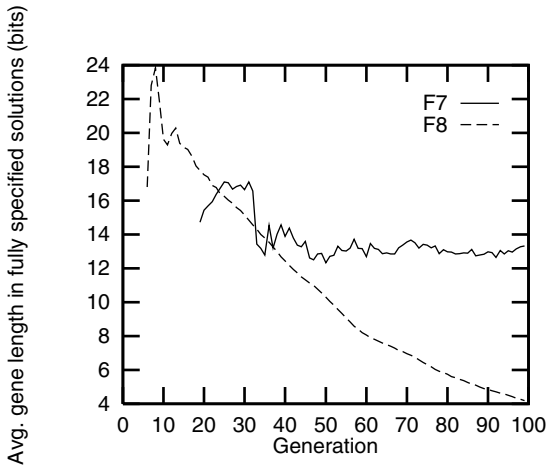


Figure 3: Average length of the genes in individuals with fully specified solutions. In the early generations none of the solutions are fully specified and no data is presented. Once the solutions are fully specified the average length of the genes decreases; steadily for F8 and quite slowly for F7.

There is another critical fact to draw from Figure 3. Consider generation 32 where the average gene length for both problems is roughly 15 bits. Including the start and stop codons each gene takes up, on average, 25 bits. Because the data is only taken from chromosomes in which all 25 variables are specified this means that the total length of the genes in one solution, measured in bits, is 625 (25 genes times 25 bits per gene). However, the lengths of the chromosomes for the F7 and F8 problems are only 250 and 400 bits respectively. The GA has succeeded in compressing a total gene length of 625 bits into 250 and 400 bits respectively.

However, it is also true that performance on F8, where there is less compression, is much better. In fact, by the final generation the average gene length for F8 is roughly 14 (including start and stop codons). This leads to a total gene length of 350, which is less than the chromosome length. Thus, although the GA can perform information compression within the multiple reading frames it seems to perform better when it does not need to.

Because the GA has found a solution (or solutions) to F8 that can be encoded in very short binary sequences, there are fewer bits per variable that need to be determined to find the solution. Whereas the standard GA must optimize all 250 bits in its chromosome, the GA with reading frames must only optimize the bits within genes. By reducing this number the GA reduces the size of the solution space and makes the problem easier. It is also possible that the genes are less susceptible to the effects of crossover and mutation as they represent a smaller percentage of the entire chromosome.

Finally, we are interested in how much overlap actually occurs between genes in different reading frames. A given bit can be read in five different reading frames. Thus, each bit can 'participate' in from 0 to 5 genes. Figure 4 shows the number of bits that participate in 0, 1, 2 or 5 genes (the values for 3 and 4 genes are omitted for clarity) for the F7 function. These results are the average of 30 trials. Each bit that participates in more than one gene represents a point of overlap. Bits where zero genes overlap are non-coding.

Initially, the majority of the bits do not participate in any gene, i.e. most of a chromosome is non-coding. Again this is because in the initial random individuals there are relatively few start-stop codon pairs. However, the number of bits that do not participate decreases extremely rapidly. There are corresponding increases in the number of bits that participate in one or more genes.

There are two important features of this figure. First, the amount of overlap is quite high. Many bits are participating in several (and often all five) genes. This explains how genes whose total length average 625 bits can be squeezed into a chromosome 250 bits long.

The second important feature of the graph is how the amount of overlap changes over time. The number of bits that participate in zero genes decreases rapidly until roughly generation 18. Generation 18 also roughly corresponds to the point when all twenty-five variables are specified. This suggests that the GA attempts to rapidly specify all of the variables. The steady

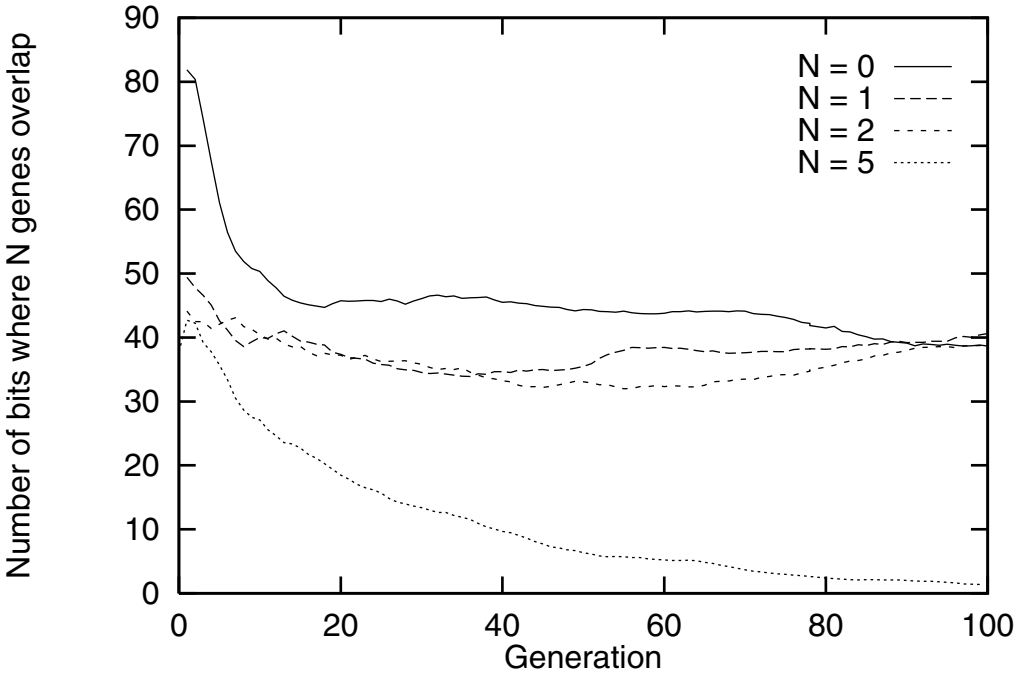


Figure 4: Number of bits with overlap and the amount of overlap (0, 1, 2 or 5 genes) at those bits, for the F7 function. The count of bits where 3 or 4 genes overlap are left out for clarity. The rapid decrease in the number of bits with 0 overlap, i.e. no gene is using them, illustrates that the chromosomes are evolving to utilize more of the available chromosome length. Fluctuations in the other values shows that gene overlap is occurring between the reading frames and that the amount of overlap appears to be subject to evolutionary pressure.

decrease in the number of bits shared by five genes suggests that the GA is also trying to separate the genes to minimize overlap. This seems reasonable, as it should be easier to optimize bits that participate in fewer genes. Thus, the overall behavior of the GA appears to be extremely sophisticated.

Figure 5 shows the number of bits participating in 0, 1, 2, or 5 genes for the F8 function (again the 3 and 4 gene cases have been omitted for clarity). The results are somewhat different from those seen with the F7 function. Except in the very earliest generations the number of non-coding bits increases throughout the trial and the number of bits participating in five genes decreases. Apparently there is much less pressure to produce overlapping genes.

However, the number of bits where two genes overlap does increase for roughly the first fifty generations and then starts to decline. Thus, it still appears that initially there is strong pressure to specify all of the variables. Later there is pressure to reduce the amount of overlap as much as possible. However, because the F8 function can apparently be solved with relatively short genes the amount of overlap never needs to be as high as for the F7 function. Additionally, the increased length of the chromosome for F8 allows less overlap even if the genes were the same length.

7 Discussion

It is clear that a GA can introduce new 'genes' as necessary to solve a given problem, even with the difficulties imposed by using start and stop codons and overlapping genes. More importantly, where necessary the GA can take advantage of multiple reading frames to overlap genes producing a form of information compression. In the most extreme case genes totaling 625 bits in length are represented in a chromosome 250 bits long. However, it is also clear that too much overlap between genes significantly degrades the performance of the GA. Thus, the potential benefits of this compression may be limited. Interestingly, after introducing sufficient genes, which often requires considerable overlap, the GA appears to reduce the amount of overlap (and hence the amount of compression) on its own.

It appears that multiple reading frames can be useful on problems where the length of the optimal solution is unknown, but the maximum possible solution length is quite large. Rather than requiring a chromosome as long as the maximum solution length, reading frames can be used to compress the chromosome. As

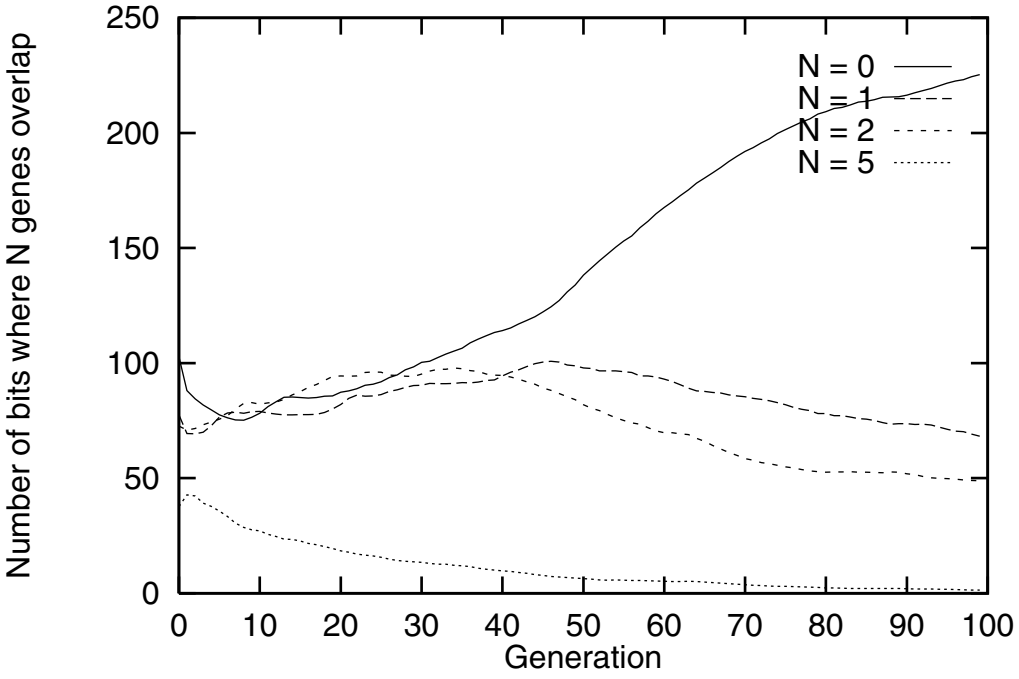


Figure 5: Number of bits with overlap and the amount of overlap (0, 1, 2 or 5 genes) at those bits, for the F8 function. The count of bits where 3 or 4 genes overlap are left out for clarity. The rapid increase in the number of bits with 0 overlap, i.e. no gene is using them, illustrates that the chromosomes are evolving very short genes. Fluctuations in the other values shows that gene overlap is occurring between the reading frames and that the amount of overlap appears to be subject to evolutionary pressure.

observed here the GA can be expected to adjust the actual length of the genes within the chromosome as necessary to solve the problem.

Results with the F3 and F3' problems emphasize a potential difficulty in using multiple reading frames. The choice of codons may make it difficult for a GA to find overlapping genes that encode the optimal solution. Without this overlap the benefits of multiple reading frames are lost. A likely solution to this problem is to increase the amount of redundancy within the encoding. Additional redundancy makes it more likely that there are ways to produce optimal genes with overlapping regions.

References

[1] H. Kargupta, "The Genetic Code and the Genome Representation," GECCO-2000 Workshop on Gene Expression, held in Las Vegas, Nevada, July 2000.

[2] H. Kargupta, "Gene: Expression: The Missing Link in Evolutionary Computation," GECCO-2000 Workshop on Gene Expression, held in Las Vegas, Nevada, July 2000.

[3] P. J. Russell, *Genetics* (Scot Foresman and Company, Glenveiw, Illinois, 1990).

[4] D. Graur and W. Li, *Fundamentals of Molecular Evolution, Second Edition* (Sinauer Associates, Sunderland, MA, 2000).

[5] A. S Wu and R. K. Lindsay, "A Comparison of the Fixed and Floating Building Block Representation in the Genetic Algorithm," *Evolutionary Computation*, 4:2, 1996.

[6] D. E. Goldberg, B. Korb and K. Deb, "Messy Genetic Algorithms: Motivation, Analysis, and First Results," in *Complex Systems*, **3**(5) (1989) 493-530.

[7] K. A. De Jong, *Analysis of Behavior of a Class of Genetic Adaptive Systems*, PhD Thesis, University of Michigan, Ann Arbor, MI, 1975.

[8] M. A. Potter, K. A. De Jong, "A cooperative Coevolutionary Approach to Function Optimization," The Third Parallel Problem Solving from Nature, held in Jerusalem, Israel, 1994.

[9] D. E. Goldberg, *Genetic Algorithms in Search Optimization, and Machine Learning* (Addison-Wesley Publishing, Reading, MA, 1989).

Verification of the Theory of Genetic Algorithm Continuation

Ravi P. Srivastava and **David E. Goldberg**
Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL - 61801
{srivasta, deg}@uiuc.edu Phone: 1-217-333-2346

Abstract

This paper makes a first attempt to study and verify empirically the theory of continuation through systematic formulation of experiments. Both the basic, and in a sense bounding, cases of building block salience, as encountered in difficult problems, are dealt with individually. Experimental results closely match theory and assure us of the usefulness of an apt blend of continuation operators with epoch-wise runs.

1 INTRODUCTION

Undoubtedly, genetic algorithms (GAs) have come a long way both in terms of a wide range of applications as well as the development of more sophisticated and advanced algorithms needed therein. As GAs are entrusted with more complex and commercially important problems, it becomes imperative to develop and verify predictive theories for increasing efficiency and economics of the algorithms. Given a sufficiently large population size, genetic algorithms yield satisfactory results for a range of problems. However, when resources and time are limited, as is the case with any real-world problem, it is imperative to use those resources judiciously.

One critical aspect of GA runs is the preservation of diversity to ensure supply of appropriate building blocks as and when the algorithm seeks to solve them. Moreover, often all the contributing building blocks (BBs) are not of equal salience, that is some of them may contribute much more to the overall fitness of a solution than others. Particularly in cases where the nature of the problem requires a primarily serial processing of building blocks from high salience to low salience, it is essential to either preserve the low-salience building blocks until the end of the run, or to re-supply them when the GA is ready to process them. Several successful attempts have been made in this and related fields which enhance the problem-solving capabilities of GAs. These include dominance and diploidy (Goldberg & Smith, 1987), adaptive and self-adaptive mutation operators (Bäck & Schwefel, 1995) and linkage learning (Harik & Goldberg, 1996; Harik, 1997).

More recently, this problem of supplying diversity temporally as needed was called the *continuation problem* (Goldberg, 1999) and a bounding theory of continuation was developed that hinged on whether the problem being solved had building blocks of comparable fitness salience. That theory considered both ideal continuation operators (ICOs) with no marginal costs of continuation and real continuation operators (RCOs) with non-zero rework. Although the original paper developed the theory in some detail, no experimental evidence was presented. This paper remedies that situation by performing bounding computations using ICOs and RCOs on problems of uniform and exponential salience.

We make systematic tests using standard trap functions which are known to be hard (Deb & Goldberg, 1993) for simple GAs to solve. Comparisons are made between GA runs which use several small *epochs* (where a smaller population is revived by injecting diversity at the end of a relatively smaller convergence epoch) to solve a problem and those which use a single large epoch to solve the problem given the same number of total function evaluations. The continuation operator models are discussed in some detail with emphasis on their design from an implementation viewpoint. Results obtained are encouraging and agree with the continuation theory.

The next section briefly reviews the continuation theory as derived elsewhere (Goldberg, 1999). Section 3 discusses the design of continuation operators and

the requirements therein. It extends the previous designs from an experimental and implementation point of view. Section 4 spells out the experimental setup and the result measurement criteria. Section 5 discusses the results obtained and their consequences. We conclude with a detailed note on future work in section 6 and a summary with implications in section 7.

2 CONTINUATION THEORY

Before delving into the modeling—implementation issues, experimental design and results of the study, we briefly review the continuation theory presented elsewhere (Goldberg, 1999) and the predicted results for continuation operators. This section looks at some of the basic assumptions of the derivations and the final results for uniform and exponentially scaled BB salience are given.

To summarize, the theory for a uniformly scaled building block problem predicts best solution quality for single-epoch runs for real continuation operators and proves quality indifference to population sizing for ideal continuation operators. On the other hand, for exponentially scaled building block salience, the theory predicts best solution quality for building-block-by-building-block solution—or in other words, a run which uses several small epochs as against a single large epoch for solving the problem, both for real and ideal continuation operators.

The theory builds on the population-sizing equation (Harik, Cantú-Paz, Goldberg, & Miller, 1997)

$$n = -c\frac{\sigma}{d}\sqrt{m}\log\alpha, \tag{1}$$

where c depends on the problem difficulty, α is the probability of not meeting criterion, d/σ is the signal to noise ratio and m is the number of building blocks; together with the primary continuation equation:

$$T = egn, \tag{2}$$

where T is the total number of function evaluations expended to gain the solution, e the number of *epochs* and g the number of generations per epoch. While equation 2 holds for an ideal continuation operator, a real continuation operator also requires a finite amount of *rework* as part of the costs incurred for continuation. The equation is then modified to include the rework as

$$T = e[gn + r], \tag{3}$$

where r is the average number of additional function evaluations required per epoch.

These equations assume nearly equal epoch sizes, which is supported by considering the relationships obtained from convergence time studies (Mühlenbein, 1992; Thierens & Goldberg, 1994). For constant intensity selection (eg. tournament and truncation selection), the convergence duration are of $O(\sqrt{l})$ and $O(l)$ for uniform and exponential (largely varying) building block salience, respectively. Thus, the previous studies have justifiably assumed equal epoch sizes. However, these relations hold best at appropriate and nearly appropriate population sizing for the problem under consideration and at very low population sizes, the takeover time would be considerably less. A constant epoch size designed according to convergence of optimal populations holds sparingly at low population sizes. This may be explained by considering that for a constant number of function evaluations T, a sufficient number of epochs are not allowed if the same high takeover time is considered to hold for small populations. This is discussed in more detail in the results section later.

We now look at the results derived for uniform and exponentially scaled building blocks.

2.1 UNIFORM SCALING

With these assumptions and assuming a uniform marginal epoch error, solution for an ICO for uniformly scaled building blocks gives

$$\alpha_e \leq \exp(-T/(gn_0)), \tag{4}$$

where α_e is the error in the e^{th} epoch and n_0 is the function of problem size, problem difficulty, building block size and signal-to-noise ratio and is therefore constant for a given problem. Introducing the rework for an RCO, the equation is modified as

$$\alpha_e \leq \exp[\frac{-T/n_0}{g + r/n}], \tag{5}$$

which indicates that the best option for an RCO is a single large implicitly parallel epoch (large n).

2.2 EXPONENTIAL SCALING

However, for an exponentially scaled BB salience case, the theory takes a different turn. For a constant reliability solution, and for a problem where λ building blocks are solved for in a particular epoch, the *quality* Q of the solution, or the proportion of correctly solved building blocks at the end of the run is derived to be

$$Q = \frac{2T}{n'g}\frac{\lambda}{2^\lambda\sqrt{\lambda}}, \tag{6}$$

where n' is again a constant for given problem and solution quality. Thus, the above equation indicates that the most economical way to solve a badly scaled problem is to do it building block-wise or in other words, in several epochs instead of a single implicitly parallel epoch.

This brief review of continuation theory prepares ground for the discussion of the continuation issues and presentation of experimental results.

3 CONTINUATION ISSUES

As with other efficiency concerns, the theory of continuation is built on three elements

1. Time budget
2. Population Sizing
3. Run or epoch duration

In the remainder of this section, each of these is briefly discussed in the light of the needs of the continuation theory.

3.1 TIME BUDGET

Time budgeting is the primary concern of continuation with the fundamental tradeoff being that between a single large implicitly parallel epoch vs. the run being distributed over several epochs in time, for a fixed number of available function evaluations. Each epoch is marked with a suitable continuation operator.

3.1.1 Ideal Continuation

An ideal continuation operator is assumed to introduce diversity into a prematurely converged population by disturbing only incorrectly converged alleles without causing any extra rework penalties. An idealized continuation operator can hence only be constructed with full knowledge of the problem solution. An extreme ICO could simply flip the incorrect alleles (in case of binary-coded GAs, or perturb them substantially otherwise) at the end of an epoch by identifying them through this problem-knowledge. The aim is essentially to introduce diversity and to explore new zones of the solution space. Although this is a theoretical idealization, and real-life computations would be far from this, ICOs serve as a stepping stone in the understanding of continuation theory.

3.1.2 Real Continuation

As mentioned before, an RCO costs the run a finite amount of rework. Building blocks which are already converged correctly may be incorrectly disturbed by the overly zealous continuation operator and may need to be re-decided. It is therefore imperative to model continuation operators which minimize this rework requirements while serving satisfactorily to provide continuation to the run.

3.2 POPULATION SIZING

Considering convergence and continuation imperatively brings in the discussion of optimal population size for a given problem. Sufficiently predictive population sizing models have been derived elsewhere (Goldberg & Rudnick, 1991; Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997). Equation 1 defines an approximate population sizing model (Harik, Cantú-Paz, Goldberg, & Miller, 1997) for convergence within a desired error bound. The continuation model, viewed from another angle involves a tradeoff decision between large and small populations for the available time budget. It gives a new face to the existing population sizing models indicating feasible solution runs for otherwise non-optimal population settings.

3.3 EPOCH DURATION

Convergence (Mühlenbein, 1992; Thierens & Goldberg, 1994) and takeover time (Goldberg & Deb, 1991) studies are equally important in modeling the continuation operators. The convergence time model gives us an upper bound on the epoch duration or in other words, for a given time budget T, it gives an estimate on the maximum number of generations in a feasible epoch; since a run with population size greater than the optimal size and multiple epochs is decidedly not advisable. Takeover time gives us a lower bound estimate for epoch duration since calling continuation operators into play without allowing the already present good genetic material to seek the solution is equally undesirable.

4 EXPERIMENTAL SETUP

This section describes in detail the systematic experimental setup used to verify the results and enumerates the important issues therein.

4.1 TEST FUNCTION

There are several studies available on the choice of suitable functions for testing GA enhancements and models (Goldberg, 1992; Goldberg, 1987; Deb & Goldberg, 1993). Of these the class of trap functions are proved to be GA hard and are particularly of interest from an experimental point of view for testing algorithm improvements. The bounding difficulty of these functions (Deb & Goldberg, 1993) lends credit to results and allows for extension to hard problems. In this study, tests for continuation operators were done for *fully deceptive* (maxtrap) functions. In particular, twenty 4-bit Liepins and Vose's functions (Liepins & Vose, 1990) were concatenated to form an eighty bit function with 20 BBs. The trap, as shown in figure 1, is defined by

$$f(x) = \begin{cases} 1 & \text{if } u(x) = l \\ 1 - \frac{1+u(x)}{l} & \text{otherwise,} \end{cases} \tag{7}$$

where l is the total length of the trap, and $u(x)$ denotes the unitation (the number of bits set to one in the trap).

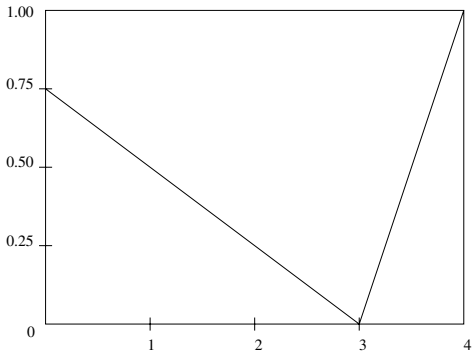


Figure 1: Single 4-bit Trap function, used as a Building Block for the 80-bit test function

For uniform salience function tests, the individual's fitness was computed as the sum of the fitnesses of the constituent BBs. For the case of non-uniform BB contribution, exponential scaling was used to vary BB salience by scaling up each constituent BB's fitness so that the signal difference is given as

$$d = 2^{1-\lambda}, \tag{8}$$

where λ is the index of BBs from high salience to low salience within the chromosome string (Goldberg, 1999). Selectorecombinative GAs were used in all runs with one-point or two-point crossover probability of 1.0 while the background mutation probability was set to zero. Tight linkage was assumed and pairwise tournament selection without replacement was used in all experiments.

4.2 MODELING ICOs

As outlined in (Goldberg, 1999) and discussed above, an ICO provides a bounding case of perfect continuation without rework costs. Exploiting the function domain information, incorrectly converged alleles alone were mutated with increased probability at the end of an epoch. Attempts were made to model an ICO such that it may be easy to extend them to a real problem where one has little or no information about the characteristics of alleles which form the global solution.

Looking at this through a slightly different lens, it may be argued that a background mutation tries to do exactly this but that is in an absolutely different flavor and context and with much lesser zeal.

4.3 MODELING RCOs

The ICO assumption may only be modeled when information on the solution is available. However, the idea may be extended to real cases under the banner of an RCO. A simplest RCO was modeled where the problem information being provided to the ICO was withdrawn. This left continuation possibly operating even on correctly converged BBs thereby introducing the rework as included in equation 3. Although this is possibly a crudest form of RCOs, it serves well as a bounding case to evaluate the theory. It represents an extreme in RCO modeling and more suggestions are made regarding *less blind* RCOs in the section on future work.

4.4 CONVERGENCE CRITERIA

In the primary runs, a convergence criterion was used based on fitness difference to determine dynamic epochs instead of specifying uniform epoch durations as assumed in the previous theory. A population was considered converged when the difference between the maximum and average individual in the population fell below the signal difference being tackled at that stage. However, runs with equal epochs tailored to corresponding population sizes were also taken and compared with theoretical predictions as well as with the dynamic-epoch model.

Results are measured in terms of the proportion of correct building blocks in the converged population after expending a fixed number of total function evaluations. From the point of view of continuation theory, it is best

suitable to compare these results for runs with varying population sizes. This gives a direct estimate of efficiency of runs for several epochs vs. a single epoch.

5 RESULTS & DISCUSSION

Runs were recorded for the two bounding cases of uniform and exponentially scaled BBs (Rudnick, 1992; Thierens, 1995) for the test function outlined in the previous section. This section presents the results for uniformly and badly scaled BBs under dynamic as well as equal epoch assumptions.

5.1 DYNAMIC-EPOCH RUNS

We first take up the results obtained when the continuation criteria was determined based on the fitness difference in the population as discussed above, rather than assuming a fixed epoch duration.

5.1.1 For Uniformly scaled BBs

Figure 2 shows the results for the two operators and trap function with uniform building block salience, comparing the ICO, RCO and standard GA runs for 15,000 function evaluations[1]. Runs with an ideal continuation operator are indifferent to the population sizing; that is, the runs with a single implicitly parallel epoch are equivalent to runs where the problem is solved with smaller populations, over several epochs. This agrees with the continuation theory for uniformly scaled BBs as given by equation 4

However, with no or little solution information—where a real continuation operator is employed—finite rework introduced according to equation 5 lowers the solution quality for small population sizes. Experimental results verify the superiority of a single-epoch run over those with several epochs. A smaller population causes considerable penalties to the run duration due to the large rework for *re-solving* those parts of the solution which have already been decided for in previous generations. Solution quality in terms of the proportion of correct building blocks found by the GA is compared with simple GA runs (with no continuation), as replicated from previous experimental studies (Harik, Cantú-Paz, Goldberg, & Miller, 1997).

5.1.2 BBs Scaled Exponentially

Figure 3 shows ICO runs for the trap function where the constituting building blocks are exponentially scaled. As expected from equation 6, runs with smaller populations, i.e. those where the solution is obtained by solving fewer building blocks at a time, are increasingly more economical compared to those with fewer epochs or large population sizes. These results are compared for three T values, 5,000 10,000 and 15,000. Such a comparison highlights the utility of continuation operators in solving hard problems economically (using overall lesser function evaluations) by employing several small run epochs.

[1] Runs with continuation operators yielded equally good results with lesser function evaluations. However, this number is chosen for easy comparison with the control case.

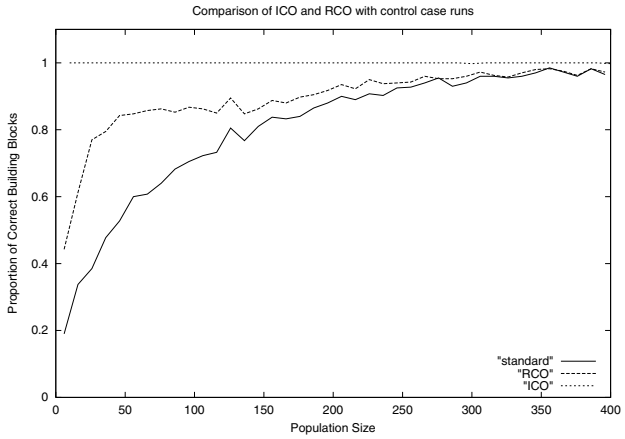


Figure 2: Proportion of correct building blocks in the solution at final convergence versus population size for uniformly scaled building block problem. Results are shown for the test trap function; comparing the ICO (dotted), RCO (dashed) and standard simple GA (solid line) runs each with 15,000 total function evaluations

5.2 EQUAL-EPOCH RUNS

Inferior solution qualities were obtained from runs under the equal-epoch assumption, possibly owing to reasons discussed in section 2. A limitingly low population converges faster than predicted by standard convergence theories.

An equal-epoch assumption continues to run a smaller population even beyond convergence, thereby reducing the total number of epochs which could otherwise be possible through a dynamic-epoch consideration.

However, if epochs of smaller sizes—suitable for low population size convergence are considered for the constant-epoch size runs, results favorably match the theoretical predictions, though not as well as in the dynamic-epoch assumption owing to the fine *tuning* requirements of the epoch size. Figure 4 shows the results for ICO runs for uniformly and exponentially scaled BBs where appropriate equal epoch sizes are

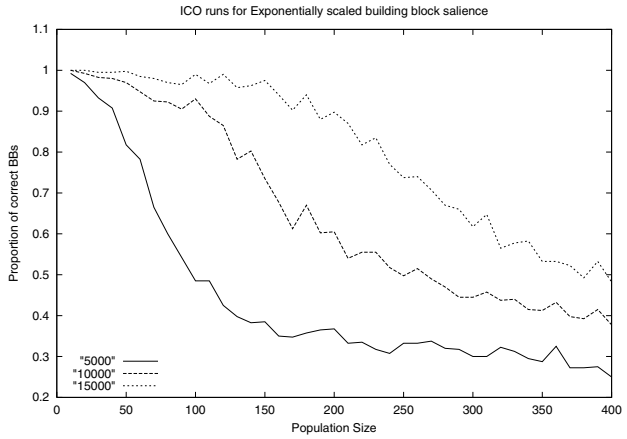


Figure 3: Proportion of correct building blocks in the solution at final convergence versus population size for exponentially scaled building block problem. Results are shown for the test trap function; comparing the ICO for 5,000 (solid), 10000 (dashed) and 15000 (dotted line) total function evaluations

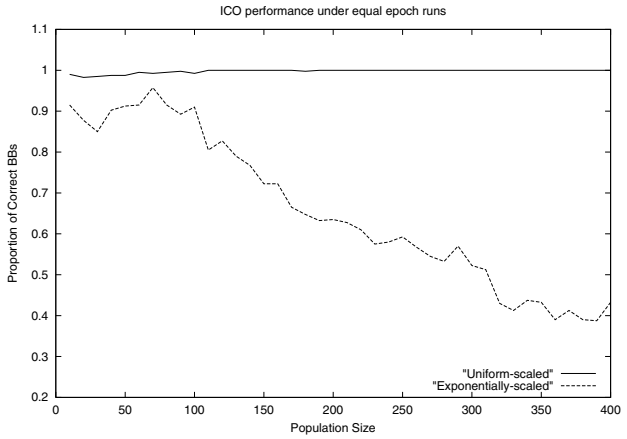


Figure 4: Proportion of correct building blocks in the solution at final convergence versus population size for uniformly (solid) and exponentially scaled (dashed) building block problem. Results are shown for the test trap function; comparing the two salience cases for 15,000 total function evaluations.

assumed for the continuation operator. Specifically, an exponentially scaled building block problem was given a constant epoch size of up-to-25 generations for smaller populations.

The experimental results confirm theoretical expectations and provide useful insight into the continuation of genetic algorithm runs in time by addressing the utility of recombination and diversity rejuvenating operators when used in the correct proportion for a given set of problem difficulty.

6 FUTURE WORK

This study verifies the effects of continuation operators, particularly depicting their usefulness in the case of hard problems (with non-uniform building block salience) by taking up the bounding case of exponentially scaled BBs.

However, several questions still remain to be answered and further new ones have sprung up during the course of the study itself. Possible directions of investigation include the following:

- **Other BB salience:** The present theory successfully predicts continuation for quantitatively distinct function regimes depending on BB salience. It is yet to be probed whether these predictions can be extended to problems of
  - mixed BB salience distribution
  - intermediate BB salience distribution

  and to what extent. The original theory touches upon this aspect by considering *neutral scaling* as one case where the quality function begins to be indifferent to variations in BB salience.
- **Other problem characteristics:** The continuation theory distinguishes between problems based on difference in BB salience alone. However, more factors may affect the algorithm's performance and should be considered.
- **The RCO spectrum:** The current theory treats RCOs as a single entity. However, evidently there exists a spectrum of RCO designs, varying from crude fixed mutations to adaptive, self-adaptive or gene expression mutation. More detailed study in the area is warranted to determine how well the theory captures this spectrum - and to extend it to capture any lacunae.
- **Design of RCOs:** It remains to be studied how the continuation operators would affect GA runs in a real setting in hard problems. This immediately spells out the need to delve into the design of RCOs themselves. RCOs with minimal rework hold the key to the success of continuation.
- **Dynamic vs. constant epochs:** The theory considers constant epoch continuation, rendering a batch-oriented cast to the runs. However, most real GAs operate continuously. This is also suggested by the dynamic-epoch experimental runs in this study. The theory may accordingly need modifications to cover this aspect.

- **Design of continuation-based efficient GAs:** How the concept of continuation may be used to design efficient GEAs is of prime importance. Continuation holds promise for optimal combines of selectorecombinative and selectomutative GAs. Systematic study to explore the current theory models may hold substantial clue to efficient GA design.

Results for RCOs for uniform building block salience clearly indicate that these operators are not mere theory but are of significant value for actual evolutionary computations. More systematic experiments need to be done in the above mentioned areas to help arrive at unified guidelines for continuation operators.

7 CONCLUSIONS

This paper verified the theory of continuation through systematic experiment formulation and tests with maxtrap functions. The ideal continuation operator was shown to be indifferent to epoch sizing for uniformly scaled BB salience while for badly scaled BBs, better solution quality was obtained for a smaller population running over several epochs under continuation. RCOs were also designed and their performance was measured. The results presented in the paper clearly express the validity of continuation theory and provide ample evidence for their usefulness in providing quality solution to hard problems economically.

Since the continuation theory is predictive, we suggest that researchers should begin making the *salience-sequence* connection. Uniform salience suggests implicit parallelism while non-uniform salience implies a necessary sequential nature to the processing. Thus the usual battle of crossover vs. mutation is increasingly being proved to be wrong-headed. Continuation strongly indicates that we should rather be thinking in terms of an optimal combine of implicitly parallel and implicitly sequential operators to get quality solutions most quickly. More work is needed and is currently underway, but this shift in perspective should help clarify the research needed and dictate the methods used in practice.

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0163. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.

We also thank Martin Pelikan and Kumara K.N. Sastry for their valuable suggestions on the contents and presentation of the material.

References

Bäck, T., & Schwefel, H.-P. (1995). Evolution strategies I: Variants and their computational implementation. In Winter, G., Périaux, J., Galán, M., & Cuesta, P. (Eds.), *Genetic Algorithms in Engineering and Computer Science* (Chapter 6, pp. 111–126). Chichester: John Wiley and Sons.

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 93–108). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing* (Chapter 6, pp. 74–88). Los Altos, CA: Morgan Kaufmann.

Goldberg, D. E. (1992). Construction of high-order deceptive functions using low-order Walsh coefficients. *Annals of Mathematics and Artificial Intelligence*, *5*, 35–48.

Goldberg, D. E. (1999). Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1* (pp. 212–219). San Francisco, CA: Morgan Kaufmann Publishers.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, *1*, 69–93. (Also TCGA Report 90007).

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D. E., & Rudnick, M. (1991). Genetic algorithms and the variance of fitness. *Complex*

Systems, *5*(3), 265–278. (Also IlliGAL Report No. 91001).

Goldberg, D. E., & Smith, R. E. (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 59–68). Hillsdale, NJ: Lawrence Erlbaum Associates.

Harik, G., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation* (pp. 7–12). Piscataway, NJ: IEEE.

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor. Also IlliGAL Report No. 97005.

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. In Belew, R. K., & Vose, M. D. (Eds.), *Foundations of Genetic Algorithms 4* (pp. 247–262). San Francisco: Morgan Kaufmann.

Liepins, G. E., & Vose, M. D. (1990). Representational issues in genetic optimization. *Journal of Experimental and Theoretical Artificial Intelligence*, *2*, 101–115.

Mühlenbein, H. (1992). How genetic algorithms really work: I.Mutation and Hillclimbing. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, 2* (pp. 15–25). Amsterdam, The Netherlands: Elsevier Science.

Rudnick, W. M. (1992). *Genetic algorithms and fitness variance with an application to the automated design of artificial neural networks.* Unpublished doctoral dissertation, Oregon Graduate Institute of Science & Technology, Beaverton, OR.

Thierens, D. (1995). *Analysis and design of genetic algorithms.* Leuven, Belgium: Katholieke Universiteit Leuven.

Thierens, D., & Goldberg, D. E. (1994). Convergence models of genetic algorithm selection schemes. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving fron Nature, PPSN III* (pp. 119–129). Berlin: Springer-Verlag.

SOME EXACT RESULTS FROM A COARSE GRAINED FORMULATION OF GENETIC DYNAMICS

C. R. Stephens
NNCP, Instituto de Ciencias Nucleares,
UNAM, Circuito Exterior, A.Postal 70-543
México D.F. 04510
e-mail: stephens@nuclecu.unam.mx

Abstract

We extend a recently developed exact schema based, or coarse grained, formulation of genetic dynamics [8, 9, 10] and its associated exact Schema theorem to an arbitrary selection scheme and a general crossover operator. We show that the intuitive"building block" interpretation of the former is preserved leading to hierarchical formal solutions of the equations that upon iteration lead to new results for the limiting distribution of a population in the case of 1-point crossover and "weak" selection, where we define quantitatively "weak". We also derive an exact, analytic form for the population distribution as a function of time for a flat landscape and 1-point crossover.

1 Introduction

Developing a better qualitative and quantitative understanding of the theory of genetic dynamics, and thereby of Genetic Algorithms (GAs), remains a challenging problem by any measure. The benefits are almost too obvious to mention. However, it is worth emphasizing the scale of the task. Almost all known theoretical results are for systems with a small number of loci and even in the case of very simple landscapes very little is known as far as explicit, rather than formal or numerical, results is concerned. In particular we are unaware of any systematic schemes for developing approximate solutions such as perturbative methods.

Passing beyond canonical results, such as Holland's Schema theorem [1], which is associated with an inequality for the dynamics rather than an equality, various exact evolution equations have been derived previously: Goldberg and Bridges [2] wrote down exact equations for two-bit problems. These equations allowed for an explicit analysis of string gains and losses. Whitley and Crabbe [3] also presented an algorithm for generating evolution equations for larger problems that was equivalent to the earlier equation of [2]. Although exact these equations are extremely unwieldy and it is difficult to infer general conclusions from their analysis.

Another related approach is that of Vose and collaborators [4] that treats GA evolution as a Markov chain. Noteworthy, and less familiar in the GA community, is analogous work in mathematical evolution theory [5] (see also the significant article of Altenberg [6] that has a foot in both camps). This formulation of GA dynamics appears to be so removed from elements such as the Schema theorem and Building Block hypothesis that the latter have very much fell out of favour with aficionados of this dynamical formulation. Also, being fundamental, microscopic equations they do not lend themselves easily to a treatment of schemata. It is important to note that this is not just an academic point: genetic dynamics with a large number of degrees of freedom is so complex that a formalism that treats the effective degrees of freedom at a more macroscopic level is certainly required. Such a formalism could be postulated directly, such as in the effective theory of Shapiro and collaborators [7], or better, derived from the underlying microscopic dynamics.

Evolution equations that offer a very intuitive interpretation, illuminate the content of the Schema theorem and the Building Block hypothesis, naturally coarse grain from string equations to schema equations, yield an interpolation between the microscopic and the macroscopic and that offer new exact results or simpler proofs of known results have been recently derived [8, 9, 10]. Originally developed for a canonical GA the basic elements have also recently been extended to Genetic Programming (GP) by Poli and

coworkers [11, 12] who have showed the utility of the formalism by deriving several interesting new results.

In this paper we generalize the formalism of [8, 9, 10], to arbitrary selection schemes and a general crossover operator showing that the basic advantages of the former are preserved. We see that evolution can be fruitfully viewed as the hierarchical construction of more and more complex building blocks. The usefulness of crossover in aiding search is governed by linkage disequilibrium coefficients associated with selection probabilities rather than gene frequencies. We consider the limit distributions of a population showing that in the long time limit an arbitrary schema reaches Robbins proportions but with a coefficient different than one. We also derive an exact, analytic form for the dynamics of a GA for a flat landscape and 1-point crossover.

2 Microscopic String Evolution Equations

In this section we recapitulate the results of [8, 9, 10], taking the opportunity to show their explicit generalization to a general crossover operator and arbitrary selection. We start with an exact evolution equation that relates string proportions, $P(c_i,t) = n(c_i,t)/n$, for a genotype c_i consisting of binary alleles, at time t to those at $t+1$, where $n(c_i,t)$ is the number of strings c_i in the population at time t and n is the population size. In the limit $n \to \infty$ the $P(c_i,t)$ give the probability distribution for the population dynamics and satisfy

$$P(c_i,t+1) = \mathcal{P}(c_i)P_c(c_i,t) + \sum_{\mathcal{C}_i} \mathcal{P}(c_j \to c_i)P_c(c_j,t) \quad (1)$$

where, for an infinite population, $P_c(c_i,t)$ is the expected proportion of strings of type c_i after selection and crossover.

The effective mutation coefficients are: $\mathcal{P}(c_i)$ which is the probability that string i remains unmutated, and $\mathcal{P}(c_j \to c_i)$, the probability that string j is mutated into string i. In the limit where the mutation probability, p_m, is uniform, $\mathcal{P}(c_i) = (1-p_m)^N$ and $\mathcal{P}(c_j \to c_i) = p_m^{d^H(i,j)}(1-p_m)^{N-d^H(i,j)}$, where $d^H(i,j)$ is the Hamming distance between the strings c_i and c_j. Note that for a finite population the left hand side of (1) is the expected proportion of genotype c_i to be found at $t+1$ while any $P(c_i,t)$ on the right hand side are to be considered as the actual proportions found at t.

Explicitly $P_c(c_i,t)$ is given by

$$P_c(c_i,t) = P'(c_i,t) - \sum_{m=1}^{2^N} p_c(m) \sum_{c_j \neq c_i} \mathcal{C}^{(1)}_{c_i c_j}(m) P'(c_i,t)P'(c_j,t)$$
$$+ \sum_{m=1}^{2^N} p_c(m) \sum_{c_j \neq c_i} \sum_{c_l \neq c_i} \mathcal{C}^{(2)}_{c_j c_l}(m) P'(c_j,t)P'(c_l,t) \quad (2)$$

where $\sum_{m=1}^{2^N}$ is the sum over all possible crossover masks $m \in \mathcal{M}$, $\mathcal{M}$ being the space of masks, and $p_c(m)$ is the probability to implement the mask m. $P'(c_i,t)$ is the probability that genotype c_i is selected and the coefficients $\mathcal{C}^{(1)}_{c_i c_j}(m)$ and $\mathcal{C}^{(2)}_{c_j c_l}(m)$, represent the probabilities that, given that c_i was one of the parents, it is destroyed by the crossover process, and the probability that given that neither parent was c_i it is created by the crossover process.

In order to write these probabilities more explicitly we denote the set of alleles inherited by an offspring from parent c_j as $\mathcal{S}$ and the alleles inherited from parent c_l, i.e. the set $c_l - \mathcal{S}$, by $\mathcal{C}$. Naturally, $\mathcal{S}$ and $\mathcal{C}$ both depend on the particular crossover mask chosen. Then,

$$\mathcal{C}^{(1)}_{c_i c_j}(m) = \theta(d^H_s(i,j))\theta(d^H_c(i,j)) \quad (3)$$

and

$$\mathcal{C}^{(2)}_{c_j c_l}(m) = \frac{1}{2}[\delta(d^H_s(i,j))\delta(d^H_c(i,l)) + \delta(d^H_c(i,j))\delta(d^H_s(i,l))] \quad (4)$$

where $d^H_s(i,j)$ is the Hamming distance between the strings c_i and c_j measured only over the set $\mathcal{S}$, with the other arguments in (3) and (4) being similarly defined. $\theta(x) = 1$ for $x > 0$ and is 0 for $x = 0$, whilst $\delta(x) = 0 \quad \forall x \neq 0$ and $\delta(0) = 1$. The equations (1) and (2) yield an exact expression for the expectation values, $n(c_i,t)$, and in the limit $n \to \infty$ yield the correct probability distribution governing the GA evolution for arbitrary selection, mutation and crossover. It takes into account exactly the effects of destruction and construction of strings and, at least at the formal level, is either a generalization of or is equivalent to other exact formulations of GA dynamics.

As an explicit example, consider 1-point crossover. In this case there are only $N-1$ pertinent crossover masks labelled by the crossover point k. Then, $p_c(m) = p_c/N-1$ for $m = k$, $k \in [1, N-1]$ and $p_c(m) = 0$ otherwise. Also, $\mathcal{S} = L$ and $\mathcal{C} = R$ (or vice versa) where L and R refer to the parts of the string to the left and right of the crossover point respectively. For 2-point crossover there are ${}^{N-1}C_2$ non-zero masks labelled by two crossover points, k_1 and k_2. Then $p_c(m) = p_c/{}^{N-1}C_2$ for $m \in \{k_1, k_2\}$ with $k_1,\ k_2 \in [1, N-1]$

and $k_1 > k_2$. In this case $\mathcal{S}$ represents the parts of the string outside k_1 and k_2 and $\mathcal{C}$ the part between them (or vice versa). As a final example, for uniform crossover $p_c(m) = p_c \mu^{N_S}(1-\mu)^{N-N_S}/2^N$ where μ is the probability that a given allele is inherited from parent c_j, $\mathcal{S}$ being the set of alleles inherited from c_j.

Computationally, the above representation is very redundant. The matrix $\mathcal{C}^{(2)}_{c_j c_l}(m)$ has dimensionality $(2^N-1)\times(2^N-1)$ but only $(2^{N_S}-1)\times(2^{N_c}-1)$ matrix elements are non-zero where N_s is the number of alleles of c_i inherited from one parent and N_c the number from the other. As a consequence, and also for other reasons, iterating the dynamics is exceedingly difficult. The equations (1) and (2) offer little in terms of intuitive understanding of the evolutionary dynamics and, arguably, not a great deal in terms of formal results, though Vose and coworkers have made important contributions in this area. Intuitive elements of GA theory such as the Building block hypothesis seem to play no role here and are not apparent in any degree. Additionally, a more general formulation giving the dynamics of schemata is relatively unnatural in this formulation.

3 Coarse Grained Evolution Equations

Far greater progress can be made by changing basis from a microscopic string basis to a coarse-grained basis associated with schemata. Such a basis emerges very naturally anyway from equation (2).

To see this, consider first the destruction term. For a given crossover mask the matrix (3) restricts the sum over the strings c_j to those that differ from c_i in at least one bit both in $\mathcal{S}$ and in $\mathcal{C}$. One can convert the sum over c_j into an unrestricted sum by subtracting off those c_j that have $d_s^H(i,j)=0$ and/or $d_c^H(i,j)=0$. We then use the fact that $\sum_{C_j} P'(c_j,t)=1$ to gain a tremendous simplification of the destruction term. Similarly one may write unrestricted sums for the construction term by subtracting off explicitly those terms that have been added. The unrestricted construction term then becomes

$$\sum_{c_j \supset c_i^{\mathcal{S}}} \sum_{c_l \supset c_i^{\mathcal{C}}} P'(c_j,t)P'(c_l,t) \tag{5}$$

where $c_i^{\mathcal{S}}$ is the part of c_i inherited from c_j, i.e. the alleles $\mathcal{S}$, while $c_i^{\mathcal{C}}$ is the part inherited from c_l, i.e. the alleles $\mathcal{C}$. Obviously, both $c_i^{\mathcal{S}}$ and $c_i^{\mathcal{C}}$ are schemata. Noting that the extra terms originating from the conversion of the restricted sums to unrestricted sums in the destruction and construction terms exactly cancel one can finally rewrite $P_c(c_i,t)$ as

$$P_c(c_i,t) = P'(c_i,t)(1-p_c) + \sum_{m=1}^{2^N} p_c(m) P'(c_i^{\mathcal{S}}(m),t) P'(c_i^{\mathcal{C}}(m),t) \tag{6}$$

where $p_c = \sum_{m=1}^{2^N} p_c(m)$ is the probability to implement crossover (irrespective of the mask) and

$$P'(c_i^{\mathcal{S}}(m),t) = \sum_{c_j \supset c_i^{\mathcal{S}}} P'(c_j,t) \tag{7}$$

and similarly for $P'(c_i^{\mathcal{C}}(m),t)$. In the absence of mutation $P_c(c_i,t) = P(c_i,t+1)$. It is important to note here that in this form the evolution equation shows that crossover explicitly introduces the idea of a schema and the consequent notion of a coarse graining relating a string c_i to its building blocks, $c_i^{\mathcal{S}}$ and $c_i^{\mathcal{C}}$, which are schemata of order N_s and $N_c = N - N_s$ respectively. Notice by going to this coarse-grained basis that the potentially $(2^N-1)\times 2^N$ destruction terms, (2^N-1) being the number of terms in the restricted string sum and 2^N being the number of possible crossover masks, have been reduced to only one term and that for a given crossover mask the $(2^N-1)^2$ construction terms have been converted into one single crossover term $P'(c_i^{\mathcal{S}}(m),t)P'(c_i^{\mathcal{C}}(m),t)$. Actually, this somewhat overstates the case as the schema averages contain (2_s^N-1) and (2_c^N-1) terms respectively. However, at the very least the change in basis has explicitly eliminated the zero elements of $\mathcal{C}^{(2)}_{c_j c_l}(m)$.

So let's recapitulate how the three different operators - mutation, crossover and selection - enter in this exact dynamics. First of all, selection is embodied in the factor P' which is a "black box" for the purposes of the dynamics. That is to say, the highly simplified and elegant form of the dynamical equation (6) is independent of the type of selection used and is a direct consequence of the form invariance of P' under a coarse graining. In other words the relationship between $P'(c_i,t)$ at the string level and $P'(c_i^{\mathcal{S}},t)$ at the coarse grained level is strictly linear, i.e. $P'(c_i^{\mathcal{S}},t) = \sum_{C_i \supset C_i^{\mathcal{S}}} P'(c_i,t)$. However, if one wishes to know how the functional dependence on P dynamically changes then one must implement a particular selection scheme. For example, with proportional selection $P'(c_i,t) = (f(c_i)/\overline{f}(t))P(c_i,t)$ where $f(c_i)$ is the fitness of string c_i and $\overline{f}(t)$ is the average population fitness. It is easy to demonstrate in this case that $P'(c_i^{\mathcal{S}},t) = (f(c_i^{\mathcal{S}},t)/\overline{f}(t))P(c_i^{\mathcal{S}},t)$, where $f(c_i^{\mathcal{S}},t)$ is the fitness of the schema $c_i^{\mathcal{S}}$, which demonstrates the form invariance of proportional selection. Thus, the form invariance in this case also

holds at the level of the Ps not just the P's. This will not generally be the case. For example with a "non-linear" selection scheme where $P'(c_i,t) = a(c_i,t)P(c_i,t) + b(c_i,t)P^2(c_i,t)$ there is no natural form invariance at the level of the Ps as $P'(c_i^s,t) \neq a(c_i^s,t)P(c_i^s,t) + b(c_i^s,t)P^2(c_i^s,t)$. We also see that the elegance of the equation (6) is independent of the type of crossover implemented. The details of the crossover mechanism enter in $p_c(m)$ and also determine which alleles are included in $\mathcal{S}$.

To iterate the equation it is clear that we need to know the dynamics of the schemata c_i^s and c_i^c. Thus we need to obtain an equivalent equation for an arbitrary schema, ξ of order N_2 and defining length l. To do this we need to sum with $\sum_{C_i \supset \xi}$ on both sides of the equation (1). This can simply be done to obtain [8, 9, 10]

$$P(\xi,t+1) = \mathcal{P}(\xi)P_c(\xi,t) + \sum_{\not\xi_i} \mathcal{P}(\not\xi_i \to \xi)P_c(\not\xi_i,t) \quad (8)$$

where the sum is over all schemata, $\not\xi_i$, that differ by at least one bit from ξ in one of the N_2 defining bits of ξ. In other words any schema competing with ξ and belonging to the same partition. $P_c(\xi,t) = \prod_{k=1}^{N_2}(1 - p_m(k))$ is the probability that ξ remains unmutated and $P_c(\not\xi_i,t) = \prod_{k\in\{\xi-\not\xi_i\}} p_m(k) \prod_{k\in\{\xi-\not\xi_i\}_c}(1-p_m(k))$ is the probability that the schema $\not\xi_i$ mutate to the schema ξ. $P_c(\xi,t) = \sum_{C_i\supset\xi} P_c(c_i,t)$ is the probability of finding the schema ξ after selection and crossover. Note the form invariance of the equation. i.e. (8) has exactly the same form as (1). To complete the transformation to schema dynamics we need the schema analog of (6). This also can be obtained by acting with $\sum_{C_i\supset\xi}$ on both sides of the equation. One obtains

$$P_c(\xi,t) = (1 - p_c\frac{N_{\mathcal{M}_r}(l,N_2)}{N_{\mathcal{M}}})P'(\xi,t) + \sum_{m\in\mathcal{M}_r(l,N_2)} p_c(m)P'(\xi^s(m),t)P'(\xi^c(m),t) \quad (9)$$

where ξ^s represents the part of the schema ξ inherited from the first parent and ξ^c that part inherited from the second. $N_{\mathcal{M}_r}(l,N_2)$ is the number of crossover masks that affect ξ, $\mathcal{M}_r$ being the set of such masks. $N_{\mathcal{M}}$ is the total number of masks with $p_c(m) \neq 0$. Obviously these quantities depend on the type of crossover implemented and on properties of the schema such as defining length.

Once again for the purposes of illustration we may consider some specific examples. For 1-point crossover: $N_{\mathcal{M}} = N-1$, $N_{\mathcal{M}_r} = l-1$, $\sum_{m\in\mathcal{M}_c} p_c(m) \to p_c/N-1\sum_{k=1}^{l-1}$, where k is the crossover point, and $\xi^s = \xi^L(k)$ the part of ξ to the left of the crossover point and $\xi^c = \xi^R(k)$ is the part to the right. Substituting into (9) one finds the results of [8, 9, 10]. For m-point crossover: $N_{\mathcal{M}} = {}^{N-1}C_m$, $N_{\mathcal{M}-r} = ({}^{N-1}C_m - {}^{N-l}C_m)$ and $\sum_{m\in\mathcal{M}_r} p_c(m) \to p_c \sum_{k_1>k_2>\ldots>k_m}$, where $k_1,\ldots,k_m$ are the m crossover points.

4 Schema Theorem

With the exact evolution equations in hand we can extend the exact Schema theorem of [8, 9, 10] to a general crossover operator and arbitrary selection scheme. Once again we state it through the concept of effective fitness [8, 9, 10]

Exact "coarse grained" Schema theorem

$$P(\xi,t+1) = \frac{f_{\text{eff}}(\xi,t)}{\bar{f}(t)}P(\xi,t) \quad (10)$$

where

$$f_{\text{eff}}(\xi,t) = \Big(\mathcal{P}(\xi)P_c(\xi,t) + \sum_{\not\xi_i}\mathcal{P}(\not\xi_i\to\xi)P_c(\not\xi_i,t)\Big)\frac{\bar{f}(t)}{P(\xi,t)} \quad (11)$$

and $P_c(\xi,t)$ is given by equation (9). The interpretation of this equation is as for its analog for 1-point crossover and proportional selection [8, 9, 10]: that schemata of higher than average effective fitness are allocated an "exponentially" increasing number of trials over time. An intuitive version of the Building Block hypothesis is part and parcel of this Schema theorem: (9) explicitly shows the competition between schema destruction and construction and shows how a schema is constructed from its building blocks ξ^s and ξ^c which in their turn are composed of building blocks $\xi^{s,s}$ and ξ^{sc} and $\xi^{c,s}$ and ξ^{cc} respectively, which in their turn... The particular building blocks depend of course on the particular mask. Note that the crossover dependent part of (9) for a given crossover mask can be written as

$$\Delta(\xi,m) = P'(\xi,t) - P'(\xi^s(m),t)P'(\xi^c(m),t) \quad (12)$$

which is a selection weighted linkage disequilibrium coefficient and, other than in a flat landscape, is a more relevant quantity than the standard linkage disequilibrium coefficient which is associated with the Ps rather than the P's. If $\Delta > 0$ then crossover has a negative effect, i.e. schema destruction dominates while if $\Delta < 0$ crossover is positive. Thus, whether the selection probabilities for the building blocks of a given schema are correlated or anticorrelated determines the nature of crossover. It is important to note that the hierarchical structure inherent in the iterative solution of (6) relating a string or

schema to its more coarse grained antecedents terminates after N_2 steps when one arrives at the maximally coarse grained effective degrees of freedom - 1-schemata. These play a priviliged role as they cannot be destroyed by crossover and in the continuous time limit satisfy $P(\xi^{(i)},t) = \exp\int_0^t((f(\xi,t')/\bar{f}(t))-1)dt'$.

One of the strengths of the present coarse grained formulation is that, as we shall see, much can be deduced simply by inspection of the basic formulas. We shall first of all put the basic equation into a yet more elegant form. We introduce a 2^N-dimensional population vector, $\mathbf{P}(t)$, whose elements are $P(C_i,t)$, $i = 1,...,2^N$. Equation (1) can then be written in the form

$$\mathbf{P}(t+1) = \overline{\mathbf{W}}\mathbf{P}_c(t) \tag{13}$$

where the mutation matrix $\overline{\mathbf{W}}$ is real, symmetric and time independent and has elements $\overline{\mathbf{W}}_{ij} = p_m^{d^H(i,j)}(1-p)^{N-d^H(i,j)}$, where for simplicity we restrict now to the case of uniform mutation (the generalization to non-uniform mutation is straightforward). Restricting attention to the case of selection schemes linear in $P(C_i,t)$, $\mathbf{P}_c(t)$ can be written as

$$\mathbf{P}_c(t) = \overline{\mathbf{F}}(t)\mathbf{P}(t) + \sum_{m=1}^{2^N} p_c(m)\mathbf{j}(m,t) \tag{14}$$

where the selection - crossover destruction matrix, $\overline{\mathbf{F}}(t)$, is diagonal, and takes into account selection and the destructive component of crossover. Explicitly, for proportional selection $\overline{\mathbf{F}}_{ii}(t) = (f(C_i)/\bar{f}(t))(1-p_c)$. Finally, the "source" matrix is given by $\mathbf{j}(m,t) = P'(C_i^s(m),t)P'(C_i^c(m),t)$. Defining the selection-crossover destruction-mutation matrix $\overline{\mathbf{W}}_s(t) = \overline{\mathbf{W}}\overline{\mathbf{F}}(t)$ we have

$$\mathbf{P}(t+1) = \overline{\mathbf{W}}_s(t)\mathbf{P}(t) + \sum_{m=1}^{2^N} p_c(m)\overline{\mathbf{W}}\mathbf{j}(m,t) \tag{15}$$

The interpretation of this equation is that $\mathbf{j}(m,t)$ is a source which creates strings (or schemata) by bringing building blocks together while the first term on the right hand side tells us how the strings themselves are propagated into the next generation, the destructive effect of crossover renormalizing the fitness of the strings.

5 Formal Solutions of the Basic Equations

Needless to say solutions of these dynamical equations are hard to come by. They represent, for binary alleles, 2^N coupled non-linear difference equations, or in the continuous time limit - differential equations. As shown in [8, 9, 10], compared to a representation based on (2), even a formal solution of (6) in the absence of mutation and for 1-point crossover and proportional selection yields much valuable qualitative information, such as a simple proof of Geiringer's theorem [13] and an extension of it to the weak selection regime. Here we extend this formal solution to the case of general crossover and mutation and for any selection scheme linear in $P(C_i,t)$. The iterated solution of (15) is

$$\mathbf{P}(t) = \prod_{n=0}^{t-1}\overline{\mathbf{W}}_s(n)\mathbf{P}(0) + \sum_{m=1}^{2^N} p_c(m)\sum_{n=0}^{t-1}\prod_{i=n+1}^{t-1}\overline{\mathbf{W}}_s(i)\overline{\mathbf{W}}\,\mathbf{j}(m,n) \tag{16}$$

The mutation matrix $\overline{\mathbf{W}}$ being real and symmetric is diagonalizable. $\overline{\mathbf{W}}_s(t)$ is not generically symmetric and therefore not diagonalizable. This solution actually holds true for arbitrary schemata. The only changes are that the vectors are of dimension 2^{N_2}, the matrices of dimension $2^{N_2}\times 2^{N_2}$, the sum over masks for the construction terms is only over the set $\mathcal{M}_r$ and that the building blocks in $j(m,t)$ are those of the schema rather than the entire string.

The interpretation of (16) follows naturally from that of (15). Considering first the case without mutation, the first term on the right hand side gives us the probability for propagating a string or schema from $t = 0$ to t without being destroyed by crossover. In other words $\prod_{n=0}^{t-1}\overline{\mathbf{W}}_s(n)$ is the Greens function or propagator for $\mathbf{P}$. In the second term, $\widehat{\mathbf{j}}(m,n)$, each element is associated with the creation of a string or schema at time n via the juxtaposition of two building blocks associated with a mask m. The factor $\prod_{i=n}^{t-1}\overline{\mathbf{W}}_s(i)$ is the probability to propagate the resultant string or schema without crossover destruction from its creation at time n to t. The sum over masks and n is simply the sum over all possible creation events in the dynamics. This formulation lends itself very naturally to a diagrammatic representation that will be discussed elsewhere. The role of mutation is to mix the strings or schemata created in the aforementioned process.

So, what can be deduced from (16)? First of all let's consider the case of a flat fitness landscape with no mutation; then $\widehat{\mathbf{P}}(t) = \mathbf{P}(t)$ and $\overline{\mathbf{W}}$ is the unit matrix, $\mathbf{1}$, and $\overline{\mathbf{W}}_s = (1-p_c)\mathbf{1}$. Then, for an arbitrary schema ξ

$$\mathbf{P}(t) = (1 - p_c\frac{N_{\mathcal{M}_r}(l,N_2)}{N_{\mathcal{M}}})^t\mathbf{P}(0)$$

$$+(1-p_c\frac{N_{\mathcal{M}_r(l,N_2)}}{N_{\mathcal{M}}})^t\sum_{m\in\mathcal{M}_r(l,N_2)}p_c(m)$$
$$\times\sum_{n=0}^{t-1}(1-p_c\frac{N_{\mathcal{M}_r(l,N_2)}}{N_{\mathcal{M}}})^{-(n+1)}\mathbf{j}(m,n) \quad (17)$$

Now, obviously $\lim_{t\to\infty}(1-p_c\frac{N_{\mathcal{M}_r(l,N_2)}}{N_{\mathcal{M}}})^t=0$ hence $\mathbf{P}(t)\to 0$ as $t\to\infty$ unless the summation over time leads to a cancellation of this damping factor. Given that the building block constituents of $\mathbf{j}(m,n)$ are associated with damping factors $(1-p_c\frac{N_{\mathcal{M}_r(l',N_2')}}{N_{\mathcal{M}}})^t$ and $(1-p_c\frac{N_{\mathcal{M}_r(l-l',N_2-N_2')}}{N_{\mathcal{M}}})^t$ this can only occur if there is no damping of the consituent building blocks and this only happens if they are 1-schemata as $N_{\mathcal{M}_r}(1,1)=0$, i.e. you can't cut a 1-schema. Given that it's a flat landscape $P(\xi^{(i)},t)=P(\xi^{(i)},0)$ where $P(\xi^{(i)},t)$ is the probability of finding the 1-schema $\xi^{(i)}$ corresponding to the bit i, hence the only term of $\mathbf{j}(m,t)$ which gives a non-zero contribution when integrated is $\prod_{i=1}^{N_2}P(\xi^{(i)},0)$. Thus the limit distribution is

$$P^*(\xi)=\lim_{t\to\infty}P(\xi,t)=\prod_{i=1}^{N_2}P(\xi^{(i)},0) \quad (18)$$

which is Geiringer's theorem for a general crossover operator.

The only role the type of crossover is playing here is how fast the transient corrections to the limit distribution die out. The damping is controlled by $N_{\mathcal{M}_r}(l,N_2)$, hence the bigger it is the faster the corresponding transient dies out. For example, for entire strings (or schemata of defining length N) for m and m'-point crossover, where $m>m'$

$$\frac{N^{(m)}_{\mathcal{M}_r}(l,N_2)}{N^{(m')}_{\mathcal{M}_r}(l,N_2)}=\left(\frac{1-\frac{\Gamma(N-l+1)\Gamma(N-m)}{\Gamma(N-l-m+1)\Gamma(N)}}{1-\frac{\Gamma(N-l+1)\Gamma(N-m')}{\Gamma(N-l-m'+1)\Gamma(N)}}\right)>1 \quad (19)$$

where $\Gamma(N)=(N-1)!$. Hence, we can quantify exactly how quickly m-point crossover transients die off relative to m'-point crossover transients.

We can take this further and generalize Geiringer's theorem to non-flat fitness landscapes for selection schemes linear in $P(c_i,t)$ such as proportional selection and for 1-point crossover and without mutation. Without loss of generality we write $P'(\xi,t)=(1+\epsilon\delta(\xi,t))$ where ϵ will serve as a control parameter to determine how "weak" selection is, $\epsilon\delta(\xi,t)$ being the selection pressure for the schema ξ. Weak selection will imply $\epsilon|\delta(\xi,t)|<1\ \forall\xi,\ t$. A 1-schema evolves as

$$P(\xi^{(i)},t)=\mathrm{e}^{\epsilon\Delta(\xi^{(i)},t,t')}P(\xi^{(i)},t') \quad (20)$$

where $\Delta(\xi^{(i)},t,t')=\int_{t'}^{t}\delta(\xi^{(i)},t'')dt''$ is the integrated selection pressure for the 1-schema.

Theorem

The limit distribution, $P(\xi)^*$, of $P(\xi,t)$ as $t\to\infty$ for "weak" selection and no mutation and 1-point crossover in the continuous time limit is

$$P(\xi)^*=\mathcal{A}(\xi,N,l,p_c)\prod_{i=1}^{N_2}P(\xi^{(i)},0) \quad (21)$$

where the selection dependent amplitude $\mathcal{A}(\xi,N,l,p_c)$ is given by

$$\mathcal{A}(\xi,N,l,p_c)=$$
$$\lim_{t\to\infty}\int_0^{\frac{p_c(l-1)t}{N-1}}\mathrm{e}^{-\tau}\mathrm{e}^{\left(1-\frac{p_c(l-1)}{N-1}\right)\Delta(\xi,t,t-\frac{N-1}{p_c(l-1)})}\times$$
$$\prod_{i=1}^{N_2}(1+\delta(\xi^{(i)},t-\tfrac{N-1}{p_c(l-1)}))\mathrm{e}^{\sum_{i=1}^{N_2}\Delta(\xi^{(i)},t-\frac{N-1}{p_c(l-1)},0)}d\tau \quad (22)$$

Note that for a flat landscape $\mathcal{A}\to 1$. Due to space limitations here a proof of this theorem will be presented elsewhere, though we can outline the logic. The key to the theorem is deciding just how weak selection must be in order that all higher order schema transients die out. It is for this reason that a tuning parameter was introduced. For a given weak selection landscape and values of l, N and p_c there exists a critical value of ϵ, $\epsilon_{cr}(\xi,N,l,p_c)$, below which all non-1-schema transients die out as $t\to\infty$. One examines the growth or decay rate of a schema and all its constituent building blocks. Growth is caused by selection and decay by crossover. By examining the dominant growth mode, other than that associated with pure 1-schemata, one can tune ϵ such that it becomes a decay mode. All other growing modes are subdominant and therefore also decay. Hence, the only non-zero mode in the limit $t\to\infty$ is the 1-schema mode of (21). The difference with the result for a flat landscape is that in this case the effective fitness landscape for 1-schemata is not flat. As an example, if one considers a weak counting ones landscape, i.e. where $f(c_i)=1+\epsilon n_1(c_i)$, $n_1(c_i)$ being the number of ones on the string, then $\epsilon_{cr}\sim p_c/N^3$ for strings.

6 Explicit Solutions

To show the further utility of the present formalism we will find the general explicit solution to (6) for a flat fitness landscape, without mutation and for 1-point crossover in the continuous time limit. Lest this be thought a trivial problem it's wise to remember that it still involves the solution of 2^N coupled non-linear

differential equations. To illustrate the general principles we'll consider first a three bit problem. The general structure of the equations (6) is that one builds up a solution via intermediate building blocks. The most fundamental blocks are 1-schemata as these cannot be cut and hence transform trivially under recombination (save in the case of uniform crossover). $\mathbf{P}^{(l,N_2)}(t)$ in this case is an N-dimensional vector and $\mathbf{P}^{(l,N_2)}(t) = \mathbf{P}^{(l,N_2)}(0)$, with $l = 1$ and $N_2 = 1$. Explicitly, $P(i**,t) = P(i**,0)$, $P(*j*,t) = P(*j*,0)$ and $P(**k,t) = P(**k,0)$. There are four 2-schemata per crossover point corresponding to $P(ij*,t)$ and $P(*jk,t)$, $i,\ j = 0,\ 1$. $P(ij*,t)$ satisfies

$$P(ij*,t) = \mathrm{e}^{-\frac{p_c t}{2}} P(ij*,0) + \frac{p_c}{2}\mathrm{e}^{-\frac{p_c t}{2}} \int_0^t P(i**,t')P(*j*,t')\mathrm{e}^{\frac{p_c t'}{2}} dt' \qquad (23)$$

with an analogous equation for $P(*jk,t)$. The simple solution of (23) is

$$P(ij*,t) = \mathrm{e}^{-\frac{p_c t}{2}} P(ij*,0) + (1-\mathrm{e}^{-\frac{p_c t}{2}})P(i**,0)P(*j*,0) \qquad (24)$$

The 3-schema solution is found using (24)

$$P(ijk,t) = \mathrm{e}^{-p_c t}P(ijk,0) + \frac{p_c}{2}\mathrm{e}^{-p_c t}\int_0^t \mathrm{e}^{p_c t'}\left(P(ij*,t')P(**k,t') + P(i**,t')P(*jk,t')\right) dt' \qquad (25)$$

Substituting the solution (24) into (25) one finds simply

$$P(ijk,t) = \mathrm{e}^{-p_c t}P(ijk,0) + \mathrm{e}^{-\frac{p_c t}{2}}(1-\mathrm{e}^{-\frac{p_c t}{2}})\left(P(ij*,0)P(**k,0) + P(i**,0)P(*jk,0)\right) + (1-\mathrm{e}^{-\frac{p_c t}{2}})^2 P(i**,0)P(*j*,0)P(**k,0) \qquad (26)$$

In the limit $t \to \infty$, $P(ijk,t) \to P(i**,0)P(*j*,0)P(**k,0)$. We see here the approach to Robbin's proportions is exponentially fast with the bigger schemata dying off quicker.

The general solution for an N-bit string is

$$P(c_i,t) = \sum_{n=0}^{N-1} \mathrm{e}^{-\frac{n p_c t}{N-1}}(1-\mathrm{e}^{-\frac{p_c t}{N-1}})^{N-n-1}\mathcal{P}(n+1) \qquad (27)$$

where $\mathcal{P}(n+1)$ is an initial condition and represents a partition over the probabilities for finding $N-n$ building blocks at $t = 0$. For a given n there are ${}^{N-1}C_n$ such terms. Equation (26) offers a simple illustration where $N = 3$, hence $n = 0,\ 1,\ 2$. $n = 0$ corresponds to the N building block terms of which there are ${}^{N-1}C_0 = 1$ term, $P(i**,0)P(*j*,0)P(**k,0)$. $n = 1$ corresponds to the $N-1$ building block terms of which there are ${}^{N-1}C_1 = 2$ terms, $P(ij*,0)P(**k,0)$ and $P(i**,0)P(*jk,0)$. Finally, $n = 2$ corresponds to the $N-2$ building block terms of which there are ${}^{N-1}C_2 = 1$ term, $P(ijk,0)$.

Notice that (27) gives not only the asymptotic behaviour but also the complete transients. We are unaware of any similar result in the literature. It is not difficult to prove that (27) is the general solution by showing that it satisfies (6). This requires the solutions of (6) for $c^L(k)$ and $c^R(k)$, the building blocks of c_i also. From the form invariance of the equations the solution of (6) is

$$P(c_i^L,t) = \sum_{n_L=0}^{l_L(k)-1} \mathrm{e}^{-\frac{n_L p_c t}{N-1}}(1-\mathrm{e}^{-\frac{p_c t}{N-1}})^{N-n_L-1}\mathcal{P}(n_L+1) \qquad (28)$$

where $N-n_L$ is the number of building blocks of $c_i^L(k)$ and $l_L(k)$ is the defining length of $c_i^L(k)$. Using the analogous equation for $c_i^R(k)$ and the fact that $n = n_L + n_R$ one sees that both sides of (6) have the same time dependence hence it suffices to prove the equivalence at $t = 0$. The equality at $t = 0$ hinges on the identity that $\mathcal{P}(n+1) = \sum_{n_L=0}^{n-1}\mathcal{P}(n_L+1)\mathcal{P}(n_R+1)$.

In the case of zero mutation the other exact limit of the evolution equations is that of pure selection. Interestingly, the solution is only exact in the case of discrete time. With these two exact limits in hand there should be scope for obtaining perturbative solutions to the evolution equations either in powers of ϵ for weak selection or in powers of p_c for strong selection.

7 Conclusions

In this article we have generalized the formalism of [8, 9, 10] to cover arbitrary selection schemes and a general crossover operator. We have tried to emphasize the advantages of this formulation and in particular show how these advantages have concrete payoffs. We believe that our coarse grained formulation is more intuitive in its content than others, giving an exact schema theorem that contains in a very obvious manner a Building Block hypothesis. Its coarse grained hierarchical structure, both in terms of time and building block complexity, leads to a formulation wherein results such as Geiringer's theorem can be seen in such a manifestly simple way that it is essentially proof by inspection. Additionally, the way that selection, schema destruction and schema creation en-

ter in different ways leads, using the same hierarchical structure, to a version of Geiringer's theorem for a class of "weak" selection landscapes where we were able to quantify the meaning of "weak".

The fundamental coarse grained equation show an important form invariance, i.e. after passing from schemata of a given N_2 to more coarse grained schemata of order $N_2' < N_2$, the resulting equations have exactly the same form. This is certainly not manifestly true of the fundamental string equations (2). This form invariance is an important property as it implies that if a solution can be found for one degree of coarse graining then an analogous solution can simply be written down for the more coarse grained degrees of freedom. We showed that the eficacy of the present formulation lies not only in its intutive appeal and the facility with which more general formal results can be deduced but also in how one may derive explicit analytic results, as in the case of an exact solution for the evolution of strings in the case of a flat fitness landscape and 1-point crossover.

We believe that our previously derived results and the results herein are the tip of the iceberg and that the present formalism may serve as a starting point for deriving a whole host of similar results and beyond. To name just a couple of obvious ones: finding the asympotic limiting distributions for other types of crossover and including in mutation and finding the exact dynamics for other types of crossover operator One of the most intriguing possibilities, that we have briefly alluded to, is the possibility of deriving approximate results using a systematic approximation scheme such as perturbation theory. The existence of exact solutions in the flat fitness landscape and the zero crossover limit add weight to such a supposition as does the fact that the iterative solution of the equations (6) leads to a diagrammatic series very similar to the Feynman diagrammatic series that appear in quantum field theory.

References

[1] Holland, J.H. (1975) *Adaptation in natural and artificial systems* (MIT Press, Cambridge, MA).

[2] Bridges, C.L. and Goldberg, D.E. (1987) "An Analysis of Reproduction and Crossover in a Binary-encoded Genetic Algorithm", in: *Genetic Algorithms and their Applications*, ed. John J. Grefenstette, (Lawrence Erbaum Publishers, Hillsdale, NJ) 9-14.

[3] Whitley, D., Das, R. and Crabb, C. (1992) "Tracking Primary Hyperplane Competitors in Genetic Search", *Annals of Mathematics and Artificial Intelligence* **6**, 367-388.

[4] Vose, M.D. and Nix, A. (1992) "Modelling Genetic Algorithms with Markov Chains", *Annals of Mathematics and Artificial Intelligence* **5**, 79-98.

[5] Karlin, S. (1979) "Models of Multifactorial inheritance: I, Multivariate formulations and basic convergence results", *Theoretical Population Biology* **15**, 308-355.

[6] Altenberg, L. (1995) "The Schema Theorem and Price's Theorem", *Foundations of Genetic Algorithms 3*, ed.s D. Whitley and M. Vose, 23-49 (Morgan Kaufmann, San Mateo).

[7] Prühgel Bennett, A. and Shapiro, J. (1994) "An analysis of genetic algorithms using statistical mechanics", *Phys. Rev. Lett.* **72**, 1305-1309.

[8] Stephens, C.R. and Waelbroeck, H. (1997), "Effective Degrees of Freedom in Genetic Algorithms and the Block Hypothesis", *Proceedings of the Seventh International Conference on Genetic Algorithms*, ed. T. Bäck (Morgan Kaufmann, San Mateo) 34-41.

[9] Stephens, C.R. and Waelbroeck, H. (1998) "Analysis of the Effective Degrees of Freedom in Genetic Algorithms", *Physical Review* **D57** 3251-3264.

[10] Stephens, C.R. and Waelbroeck, H. (1999) "Schemata Evolution and Building Blocks", *Evol. Comp.* **7(2)** 109-124.

[11] Poli, R. (1999) "Schema theorems without expectations", *Proceedings of GECCO99*, ed. W. Banzhaf *et al* (Morgan Kaufmann).

[12] Poli, R. (2000) "Hyperschema theory for GP with one point crossover, building blocks and some new results in GA theory", *Genetic Programming, Proceedings of Euro GP 2000*, ed. R. Poli, W. Banzhaf *et al* (Springer Verlag).

[13] Geiringer, H. (1944) "On the Probability Theory of Linkage in Mendelian Heredity", *Annals of Mathematical Statistics* **15**, 25-27.

Using Genetic Algorithms for Learning Clauses in First-Order Logic

Alireza Tamaddoni-Nezhad
Department of Computer Science
University of York, York, YO10 5DD, UK
alireza@cs.york.ac.uk

Stephen Muggleton
Department of Computer Science
University of York, York, YO10 5DD, UK
stephen@cs.york.ac.uk

Abstract

A framework for combining first-order concept learning with Genetic Algorithms is introduced. This framework includes: 1) a novel binary representation for clauses 2) task-specific genetic operators 3) a fast evaluation mechanism. The proposed binary representation encodes the refinement space of clauses in a natural and compact way. It is shown that essential operations on clauses such as unification and anti-unification can be done by simple bitwise operations (e.g. and/or) on the binary encoding of clauses. These properties are used for designing task-specific genetic operators. It is also shown that by using these properties individuals can be evaluated at genotype level without mapping them into corresponding clauses. This replaces the complex task of evaluating clauses, which usually needs repeated theorem proving, by simple bitwise operations. An implementation of the proposed framework is used to combine Inverse Entailment of the learning system CProgol with a genetic search.

1 INTRODUCTION

In concept learning problems, there is a trade-off between the expressive power of the representation and the complexity of hypotheses search space. In the case of first-order concept learning this search space grows combinatorially and the search is usually intractable. On the other hand, many real-world applications require a representation language which is at least as expressive as first-order logic. For example, Inductive Logic Programming (ILP) (Muggleton, 1991; Muggleton and Raedt, 1994) has been a fast growing research area in the last decade and has shown many successes in machine learning and data-mining applications, especially in some challenging domains such as bioinformatics (Muggleton et al., 1992; Srinivasan et al., 1997; Muggleton, 1999). Current first-order learning systems mostly employ deterministic search methods to examine the refinement space of clauses and use different kind of syntactic biases and heuristics (e.g. greedy methods) to cope with the complexity of the search which otherwise is intractable. Using these biases usually limits the exploration power of the search and may lead to local optima. Hence, more powerful search methods are required for inducing complex concepts and for dealing with massive data. Genetic Algorithms (GAs) have great potential for this purpose. GAs are multi-point search methods (and less sensitive to local optima) which can search through a large space and manipulate symbolic as well as numerical data. Moreover, because of their robustness and adaptive characteristics, GAs are suitable methods for optimization and learning in many real world applications (Goldberg, 1989). In terms of implementation, GAs are highly parallel and can be easily implemented in parallel and/or distributed models. However, GAs are syntactically restricted and cannot represent a priori knowledge that already exists about the domain. On the other hand, first-order concept learning methods, such as ILP, are well known paradigms that benefit from the expressive power inherited from logic and logic programming. Hence, it is likely that a combination of such learning paradigms and GAs can overcome the limitation of each individual method and can be used to cope with some complexities of real-world applications.

Even though GAs have been used widely for optimization and learning in many domains, a few genetic-based systems in first-order domain already exist. Some of these systems (Giordana and Sale, 1992; Giordana and Neri, 1996; Hekanaho, 1998; Anglano et al.,

1998) follow conventional genetic algorithms and represent problem solutions by fixed length bit-strings. Other systems (Varšek, 1993; Leung and Wong, 1995; Wong and Leung, 1997; Kennedy and Giraud-Carrier, 1999; Reiser and Riddle, 1998) use a hierarchical representation and evolve a population of logic programs in a Genetic Programming (GP) (Koza, 1991) manner. These genetic-based systems confirm that genetic algorithms and evolutionary computing can be interesting alternatives for learning first-order concepts from examples. However, these systems mostly use genetic search as the only learning mechanism in the system and hence they cannot benefit from the existing first-order learning techniques for example for utilizing background knowledge in the learning process.

This paper aims to present a framework for combining first-order concept learning with GAs by introducing a novel binary encoding for clauses, relevant genetic operators and a fast evaluation mechanism. In this framework, unlike other genetic-based systems, representation and operators can be interpreted in well known first-order logic terms such as subsumption, unification and anti-unification (Plotkin, 1969; Nienhuys-Cheng and de Wolf, 1997). This representation and its interpretations and properties are introduced in the next section. Section 3 shows how some properties of the binary representation can be used to design task-specific genetic operators. In this section we show how essential operations on clauses can be done by simple bitwise operations (e.g. and/or) on binary strings. As another property of the proposed representation we show that evaluating a clause, which is a complex task, can also be done by simple and fast bitwise operations. This evaluation mechanism is introduced in section 4. Section 5 describes an implementation of the proposed framework for combining Inverse Entailment in CProgol (Muggleton, 1995) with a genetic algorithm. Evaluations and related works are discussed in section 6. Finally, section 7 summarizes the results and concludes this paper.

2 REPRESENTATION AND ENCODING

Every application of GAs requires formulating problem solutions in such a way that they can be processed by genetic operators. It has been suggested that the binary representation is a suitable coding for representing problem solutions in GAs (Holland, 1975; Goldberg, 1989). The lack of a proper (binary) representation, and consequently difficulties for definition and implementation of genetic operators, has been the main problem for applying GAs in first-order domain.

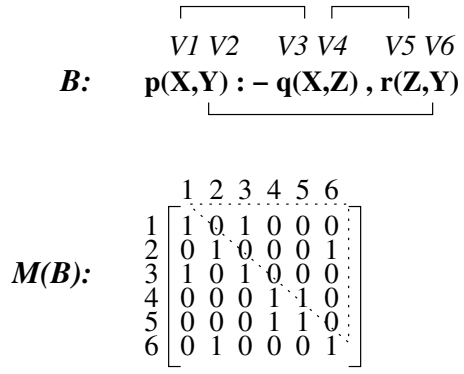


Figure 1: Binding matrix for clause *p(X,Y):-q(X,Z),r(Z,Y).*

In this section we introduce a novel binary representation for first-order clauses. We show that this representation encodes the refinement space of clauses in a natural and compact way. Even though all definition and theorems in this paper hold for the general form of firs-order clauses, for simplicity and consistency reasons, in all examples we use Horn clauses [1] which is the standard representation in logic programming. Consider a clause with n variable occurrences. The relationships between these n variable occurrences can be represented by a graph having n vertices in which there exists an edge between vertices v_i and v_j if ith and jth variable occurrences in the clause represent the same variable. For example variable bindings in clause *p(X,Y):-q(X,Z),r(Z,Y)* represent an undirected graph and this clause can be represented by a binary matrix as shown in Figure 1. In this matrix entry m_{ij} is 1 if ith and jth variable occurrences in the clause represent the same variable and m_{ij} is 0 otherwise. This representation has interesting properties which can be exploited by a genetic algorithm for searching the refinement space of a clause. Before we formally show these properties, we need to show the mappings between clauses and binary matrices.

Definition 1 (Binding Set) *Let B and C both be clauses. C is in binding set $\mathcal{B}(B)$ if there exists a variable substitution [2] θ such that $C\theta = B$.*

In Definition 1, the variables in B induce a set of equivalence classes over the variables in any clause

[1] In Horn clauses all clauses contain at most one positive literal. For example $\{p(X,Y) \vee \sim q(X,Z) \vee \sim r(Z,Y)\}$ is a Horn clause and can be represented by *p(X,Y):-q(X,Z),r(Z,Y).*

[2] Substitution $\theta = \{v_i/u_j\}$ is a variable substitution if all v_i and u_j are variables.

$C \in \mathcal{B}(B)$. Thus we could write the equivalence class of u in variable substitution θ as $[v]_u$, the set of all variables in C such that v/u is in θ. We define a binary matrix which represents whether variables v_i and v_j are in the same equivalence class or not.

Definition 2 (Binding Matrix) *Suppose B and C are both clauses and there exists a variable substitution θ such that $C\theta = B$. Let C have n variable occurrences representing variables $\langle v_1, v_2, \ldots, v_n \rangle$. The binding matrix of C is an $n \times n$ matrix M in which m_{ij} is 1 if there exist variables v_i, v_j and u such that v_i/u and v_j/u are in θ and m_{ij} is 0 otherwise. We write $M(v_i, v_j) = 1$ if $m_{ij} = 1$ and $M(v_i, v_j) = 0$ if $m_{ij} = 0$.*

Definition 3 (Normalized Binding Matrix) *Let M be an $n \times n$ binary matrix. M is in the set of normalized binding matrices $\mathcal{M}_n$ if M is symmetric and for each $1 \leq i \leq n$, $1 \leq j \leq n$ and $1 \leq k \leq n$, $m_{ij} = 1$ if $m_{ik} = 1$ and $m_{kj} = 1$.*

Definition 4 (Mapping Function M(C)) *The mapping function $M : \mathcal{B}(B) \rightarrow \mathcal{M}_n$ is defined as follows. Given clause $C \in \mathcal{B}(B)$ with n variable occurrences representing variables $\langle v_1, v_2, \ldots, v_n \rangle$, $M(C)$ is an $n \times n$ binary matrix in which m_{ij} is 1 if variables v_i and v_j are identical and m_{ij} is 0 otherwise.*

Definition 5 (Mapping Function C(M)) *The mapping function $C : \mathcal{M}_n \rightarrow \mathcal{B}(B)$ is defined as follows. Given a normalized $n \times n$ binding matrix M, $C(M)$ is a clause in $\mathcal{B}(B)$ with n variable occurrences $\langle v_1, v_2, \ldots, v_n \rangle$, in which variables v_i and v_j are identical if m_{ij} is 1.*

Definition 6 (Matrix Subset) *Let P and Q be in $\mathcal{M}_n$. It is said that $P \subseteq Q$ if for each entry $p_{ij} \in P$ and $q_{ij} \in Q$, p_{ij} is 1 if q_{ij} is 1. $P = Q$ if $P \subseteq Q$ and $Q \subseteq P$. $P \subset Q$ if $P \subseteq Q$ and $P \neq Q$.*

Definition 7 (Clause Subsumption) *Clause C subsumes clause D, $C \succeq D$ if there exists a (variable) substitution θ such that $C\theta \subseteq D$ (i.e. every literal in $C\theta$ is also in D). C properly subsumes D, $C \succ D$ if $C \succeq D$ and $D \not\succeq C$.*

Because of the subsumption order between clauses (which is a quasi-order) the search space (or refinement space) can be modeled as a subsumption lattice (Nienhuys-Cheng and de Wolf, 1997). The following theorem represents the relationship between binary matrices and the subsumption order of clauses.

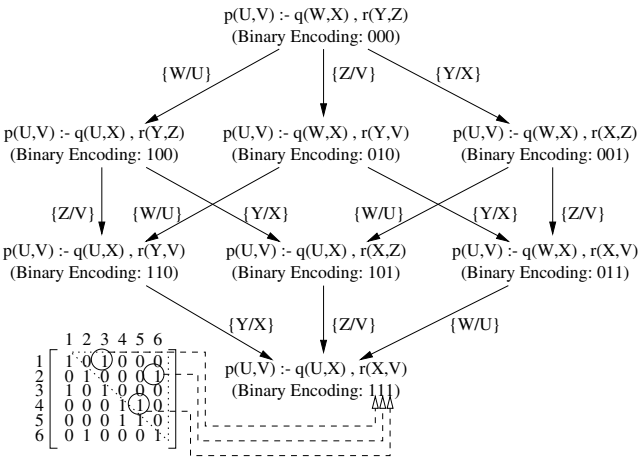


Figure 2: A subsumption lattice bounded below by clause *p(X,Y):-q(X,Z),r(Z,Y)* and binary encoding for each clause.

Theorem 1 *For each clause B and matrices M_1 and M_2 in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$ and $C(M_2) \in \mathcal{B}(B)$, $C(M_1) \succ C(M_2)$ if $M_1 \subset M_2$.*

Proof. Suppose $M_1 \subset M_2$. Therefore there exist variables v_i and v_j, $i < j$ such that $M_1(v_i, v_j) = 0$ and $M_2(v_i, v_j) = 1$. Then according to Definition 2, $C(M_1)\{v_i/v_j\} = C(M_2)$. Hence, $C(M_1) \succ C(M_2)$. □

A binding matrix is a symmetric matrix in which diagonal entries are 1. In practice, we only maintain entries in top (or down) triangle of the matrix. Furthermore, in our implementation (see section 5), we are interested in a subsumption lattice bounded below by a particular clause. Hence, each member of $\mathcal{B}(B)$ can be encoded by a binary string in which each bit corresponds to a 1 entry of matrix $M(B)$.

Example 1 *Figure 2 shows a subsumption lattice bounded below by the clause p(X,Y):-q(X,Z),r(Z,Y). Each clause in this search space can be encoded by 3 bits.*

3 GENETIC OPERATORS AND STOCHASTIC REFINEMENT

Genetic operators introduce new individuals into population by randomly changing or combining the genotype of best-fit individuals during an evolutionary process. In conventional genetic algorithms these operators are domain-independent and usually without any assumption about the problem on hand. However, more efficient genetic operators can be designed by

using simple facts about the domain. For example it has been shown that introducing generalization and specialization crossover operators, which are used together with standard crossover and mutation operators, can be useful in concept learning problems (Giordana and Sale, 1992; Janikow, 1993). In this section we show that the proposed binary representation has great potential for designing task-specific genetic operators. In particular, we show how mgi(most general instance) and lgg(least general generalization) which are also known as unification and anti-unification operations on clauses (Plotkin, 1969; Nienhuys-Cheng and de Wolf, 1997) can be achieved by simple bitwise operations on the binary encoding of clauses. In the following, first we introduce *mgi* and *lgg* operations for clauses.

Definition 8 (mgi and lgg) *Clauses E and F are respectively a common instance and a common generalization of clauses C and D if and only if $C, D \succeq E$ and $F \succeq C, D$. $mgi(C, D)$ and $lgg(C, D)$ are the most general instance and the least general generalization for clauses C and D if and only if for every common instance E and common generalization F it is the case that $mgi(C, D) \succeq E$ and $F \succeq lgg(C, D)$.*

Example 2 *In Figure 2 clause p(U,V):-q(U,X),r(X,Z) is the mgi of clauses p(U,V):-q(U,X),r(Y,Z) and p(U,V):-q(W,X),r(X,Z) and clause p(U,V):-q(W,X),r(Y,V) is the lgg of clauses p(U,V):-q(U,X),r(Y,V) and p(U,V):-q(W,X),r(X,V).*

Definition 9 (Matrix AND) *Let M_1 and M_2 be in $\mathcal{M}_n$. $M = (M_1 \wedge M_2)$ is an $n \times n$ matrix and for each $a_{ij} \in M$, $b_{ij} \in M_1$ and $c_{ij} \in M_2$, $a_{ij} = 1$ if $b_{ij} = 1$ and $c_{ij} = 1$ and $a_{ij} = 0$ otherwise.*

Similar to AND operator, OR operator $(M_1 \vee M_2)$ is constructed by bitwise OR-ing of M_1 and M_2 entries.

Definition 10 (Matrix OR) *Let M_1 and M_2 be in $\mathcal{M}_n$. $M = (M_1 \vee M_2)$ is an $n \times n$ matrix and for each $a_{ij} \in M$, $b_{ij} \in M_1$ and $c_{ij} \in M_2$, $a_{ij} = 1$ if $b_{ij} = 1$ or $c_{ij} = 1$ and $a_{ij} = 0$ otherwise.*

Theorem 2 *For each clause B and matrices M_1, M_2 and M in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$, $C(M_2) \in \mathcal{B}(B)$ and $C(M) \in \mathcal{B}(B)$, $C(M) = lgg(C(M_1), C(M_2))$ if $M = (M_1 \wedge M_2)$.*

Proof. Suppose $M = M_1 \wedge M_2$. Therefore $M \subset M_1$ and $M \subset M_2$ and according to Theorem 1 $C(M) \succ C(M_1)$ and $C(M) \succ C(M_2)$. Therefore $C(M)$ is a common generalization of $C(M_1)$ and $C(M_2)$. We show that $C(M)$ is the least general generalization of $C(M_1)$ and $C(M_2)$. For each binding matrix M' in $\mathcal{M}_n$ it must be the case that if $C(M') \succ C(M_1)$ and $C(M') \succ C(M_2)$ then $C(M') \succ C(M)$. Suppose $C(M') \not\succ C(M)$ then according to Theorem 1 there exist u and v such that $M'(u, v) = 1$ and $M(u, v) = 0$. If $M'(u, v) = 1$ then $M_1(u, v) = 1$ and $M_2(u, v) = 1$ and this contradicts $M(u, v) = 0$ and completes the proof. □

By a similar proof it can be shown that the result of or-operator is equivalent to *mgi*.

Theorem 3 *For each clause B and matrices M_1, M_2 and M in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$, $C(M_2) \in \mathcal{B}(B)$ and $C(M) \in \mathcal{B}(B)$, $C(M) = mgi(C(M_1), C(M_2))$ if $M = M_1 \vee M_2$.*

Proof. Symmetric with proof of Theorem 2. □

Example 3 *In Figure 2, lgg and mgi of any two clauses can be obtained by AND-ing and OR-ing of their binary strings.*

According to these theorems unification and anti-unification can be done by simple bitwise operations on the binary encoding of clauses. These properties can be used for designing task-specific genetic operators such as generalization and specialization crossover operators. Generalization and specialization are known as the main operations in concept learning methods (Winston, 1970; Mitchell, 1982; Mitchell, 1997). In particular, *lgg* and *mgi* are essential in first-order learning. For example, the ILP system Golem (Muggleton and Feng, 1990) which was successfully applied to a wide range real-world applications (Bratko et al., 1991; Feng, 1992; Muggleton et al., 1992) only uses a *lgg* operator which operates on determinacy restricted clauses [3]. In addition to the generalization and specialization crossovers mentioned earlier, we can also introduce task-specific mutation operators. In the standard mutation operator we use a fixed probability (mutation-rate) for changing 0 and 1 bits [4]. As shown in the previous section, the difference between bits in binding matrices determines the subsumption order between clauses. Hence, the subsumption distance between clauses increases monotonically with the Ham-

[3] It has been shown that the determinacy restriction is inappropriate in some applications (Muggleton, 1994). There is not such a restriction for the *lgg* operator introduced in this paper.

[4] A random mutation could result in a matrix which is not consistent with Definition 3. Even though this inconsistency doesn't affect the genetic search in practice, it could be avoided by a normalization closure using Definition 3.

ming distance between corresponding matrices. We can use this property to set different mutation rates for 0 and 1 bits based on a desirable degree of generalization and specialization. This leads to a directed mutation operator.

In first-order concept learning, upward and downward refinement operators are used for generalization and specialization of clauses (Nienhuys-Cheng and de Wolf, 1997). In our case, task-specific genetic operators can be interpreted as *stochastic refinement operators* in the context of first-order concept learning.

4 FAST EVALUATION AT GENOTYPE LEVEL

The usual way for evaluating a hypothesis in first-order concept learning systems is to repeatedly call a theorem prover (e.g. Prolog interpreter) on training examples to find out positive and negative coverage of the hypothesis. This step is known to be a complex and time-consuming task in first-order concept learning. In the case of genetic-based systems this situation is even worse, because we need to evaluate a population of hypotheses in each generation. This problem is another important difficulty when applying GAs in first-order concept learning.

In this section we introduce a method which replaces the complex task of evaluating clauses by bitwise operations on binary strings. This idea is similar to data-compilation method used by the attribute-based learning system GIL (Janikow, 1993). This system retains binary coverage vectors for all possible features (attributes and values) which can appear in a rule. This introduces a database which can be used for computing the coverage set of each rule by bitwise operations on the coverage vectors of participating features. However, in our case there is no need to maintain such a database. We show that by maintaining the coverage sets for a small number of clauses and by doing bitwise operations we can compute the coverage for other binary strings without mapping them into the corresponding clauses. This property is based on the implicit subsumption order which exists in the binary representation. In the following, first we define the cover-vector approach for representing coverage of a clause on training examples.

Definition 11 (Cover Sets and Cover Vectors) *Let C be a clause and $E^+ = \{e_1^+, e_2^+, \ldots, e_k^+\}$ and $E^- = \{e_1^-, e_2^-, \ldots, e_l^-\}$ be the set of positive and negative training examples respectively. e_i^+ is in the positive cover set $\mathcal{P}(C)$ if $C \succeq e_i^+$. Similarly, e_j^- is in the negative cover set $\mathcal{N}(C)$ if $C \succeq e_j^-$. The positive cover vector $\mathcal{PV}(C)$ is a k-bit binary string in which bit i is 1 if $e_i^+ \in \mathcal{P}(C)$ and 0 otherwise. Similarly, the negative cover vector $\mathcal{NV}(C)$ is a l-bit binary string in which bit j is 1 if $e_j^- \in \mathcal{N}(C)$ and 0 otherwise.*

Theorem 4 *For each clause C_1 and C_2, $\mathcal{P}(mgi(C_1, C_2)) = \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$.*

Proof. Let $e \in \mathcal{P}(mgi(C_1, C_2))$, then according to Definition 11, $mgi(C_1, C_2) \succeq e$. But according to the definition of mgi, $C_1 \succeq mgi(C_1, C_2)$ and $C_2 \succeq mgi(C_1, C_2)$ and therefore $C_1 \succeq e$ and $C_2 \succeq e$. Hence, $e \in \mathcal{P}(C_1)$ and $e \in \mathcal{P}(C_2)$ and therefore $e \in \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$. Hence, $\mathcal{P}(mgi(C_1, C_2)) \subset \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$. Now, let $e \in \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$, then according to Definition 11, $C_1 \succeq e$ and $C_2 \succeq e$. But according to the definition of mgi, $mgi(C_1, C_2) \succeq e$. Hence, $e \in \mathcal{P}(mgi(C_1, C_2))$ and therefore $\mathcal{P}(C_1) \cap \mathcal{P}(C_2) \subset \mathcal{P}(mgi(C_1, C_2))$ and this completes the proof. □

Theorem 5 *For each M, M_1 and M_2 in $\mathcal{M}_n$, $\mathcal{PV}(C(M)) = \mathcal{PV}(C(M_1)) \wedge \mathcal{PV}(C(M_2))$ if $M = M_1 \vee M_2$.*

Proof. Suppose $M = M_1 \vee M_2$. Therefore according to Theorem 3, $C(M) = mgi(C(M_1), C(M_2))$. Then according to Theorem 4 ,$\mathcal{P}(C(M)) = \mathcal{P}(C(M_1)) \cap \mathcal{P}(C(M_2))$. But according to Definition 11, $\mathcal{PV}(C(M)) = \mathcal{PV}(C(M_1)) \wedge \mathcal{PV}(C(M_2))$. □

This theorem, which also holds for negative coverage vectors, can be easily extended for n clauses. According to these theorems, positive (or negative) coverage of clauses can be computed by bitwise operations. Hence, the evaluation of each individual is done at genotype level without mapping it into the corresponding phenotype (clause).

Example 4 *In Fig. 2, $\mathcal{PV}(C(111)) = \mathcal{PV}(C(110)) \wedge \mathcal{PV}(C(101))$.*

5 IMPLEMENTATION

In our first attempt, we employed the proposed representation to combine Inverse Entailment in CProgol4.4 with a genetic algorithm. CProgol is an Inductive Logic Programming (ILP) system which develops first-order hypotheses from examples and background knowledge. CProgol uses Inverse Entailment (Muggleton, 1995) to construct the most specific clause (or the bottom-clause) for each example and then searches for the best clause H which subsumes this bottom-clause ($\square \succeq H \succeq \perp$). This introduces a subsumption lattice

bounded below by the bottom-clause ($\perp$). The standard CProgol starts from the empty clause ($\square$) and uses an A^*-like algorithm for searching this bounded subsumption lattice. The details for CProgol's A^*-like search, refinement operator and algorithm for building the bottom-clause can be found in (Muggleton, 1995).

As shown in section 2, $\mathcal{B}(\perp)$ represents a subsumption lattice bounded below by the bottom-clause. We used a genetic algorithm together with the binary encoding for clauses (as described in section 2) to evolve a randomly generated population of binary strings in which each individual corresponds to a member of $\mathcal{B}(\perp)$. Because of simple representation and straightforward operators any standard genetic algorithm can be used for this purpose. We used a *Simple Genetic Algorithm(SGA)* (Goldberg, 1989) and modified it to suit the representation introduced in this paper. This genetic search evolves a population of hypotheses which all subsume the bottom-clause and uses an evaluation function which is similar to one used in the A^*-like search of CProgol [5] but normalized between 0 and 1. The predicates which violate the mode declaration language are considered as inactive predicates which can be filtered from the induced hypothesis.

In our first experiment, we applied the genetic search to learn Michlaski's east-bound trains (Michalski, 1980). Figure 3 compares the performance of the genetic search with a random search in which each population is generated randomly as in the initial generation. In all experiments (10/10) a correct solution was discovered by the genetic search before generation 20 (standard deviations for the average fitness mean over 10 runs are shown as error bars). The following parameter setting was used for SGA: $popsize = 30$, $p_m = 0.0333$ and $p_c = 0.6$. Figure 3 also compares the convergence of the standard SGA with a SGA which uses together with standard one-point crossover, the task-specific operators lgg introduced in section 3. The following parameter setting was used for $SGA + lgg$: $popsize = 30$, $p_m = 0.0333$, $p_c = 1 - \alpha * f$ and $p_{lgg} = \alpha * f$ where f is the mean value of the fitness of the parental strings ($f = \frac{f(s_1)+f(s_2)}{2}$) and $\alpha = 0.8$ [6].

Preliminary results show that the A^*-like search exhibits better performance in learning hypotheses with small and medium complexities. However, the performance of the genetic search is less dependent on the complexity of hypotheses, whereas A^*-like search shows a great dependency on this factor. Moreover, genetic search can find the correct solution for some special cases which the solution is beyond the exploration power of the A^*-like search due to its incompleteness (Muggleton, 1995; Badea and Stanciu, 1999).

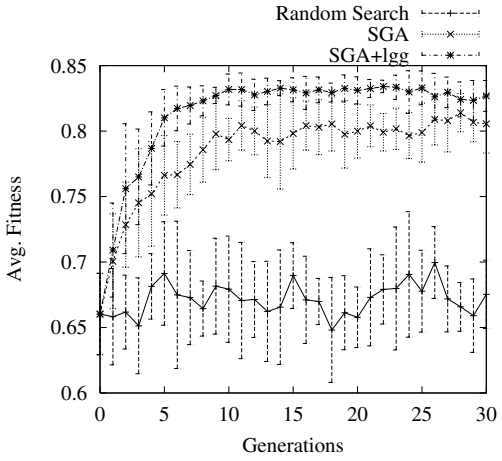


Figure 3: Convergence of the genetic search in the trains problem.

6 DISCUSSION AND RELATED WORKS

The actual completeness and complexity exhibited by the standard A^*-like search of CProgol depends upon the order of literals in the bottom clause and upon the complexity of the hypothesis. In contrast, the genetic search is less dependent on the complexity of the hypothesis and is not affected by the order of literals in the bottom clause. Therefore it is reasonable that genetic algorithm is able to find some solutions which are beyond the exploration power of the A^*-like search.

As mentioned earlier, one main difficulty in order to apply GAs in first-order domain concerns formulating first-order hypotheses into bit-strings. GA-SMART (Giordana and Sale, 1992) was the first relation learning system which tackled this problem by restricting concept description language and introducing a language template. A template in GA-SMART is a fixed length CNF formula which must be defined by the user. Mapping a formula into bit-string is done by setting the corresponding bits to represent the occurrences of predicates in the formula. The main problem of this method is that the number of conjuncts in the template grows combinatorially with the number of predicates. REGAL (Giordana and Neri, 1996),

[5] The criteria used in the evaluation function of the A^*-like search of CProgol include: number of positive and negative examples covered by the clause, length of the clause and number of further literals to complete the clause. More details can be found in (Muggleton, 1995).

[6] This parameter setting is similar to one used in (Giordana and Sale, 1992).

DOGMA (Hekanaho, 1998) and G-NET (Anglano et al., 1998) mainly follow the same idea of GA-SMART and employ a user-defined template for mapping first-order rules into bit strings. However, instead of using a standard representation, a template in these systems is a conjunction of internally disjunctive predicates. This leads to some difficulties for example for representing continuous attributes. Other systems including GILP (Varšek, 1993), GLPS (Leung and Wong, 1995), LOGENPRO (Wong and Leung, 1997), STEPS (Kennedy and Giraud-Carrier, 1999) and EVIL (Reiser and Riddle, 1998) use hierarchical representations rather than using fixed length bit-strings. These systems evolve a population of logic programs in a Genetic Programming (GP) (Koza, 1991) manner. Even though some of the above mentioned systems use background knowledge for generating initial population or seeding the population, most of these systems cannot benefit from intentional background knowledge as it is used in usual first-order learning systems. On the other hand, in our proposed framework, encoding of hypotheses is based on a most specific or bottom-clause which is constructed according to the background knowledge and training examples. This bottom-clause can be automatically constructed using logic-based methods such as Inverse Entailment. Moreover, as shown in section 2 and section 3, the proposed encoding and operators can be interpreted in well known first-order logic terms.

7 CONCLUSIONS AND FURTHER WORK

In this paper we have introduced a framework for combining first-order concept learning with GAs. Efficient binary representation for encoding clauses and its properties, relevant task-specific operators and the fast evaluation mechanism are the major novelty of the proposed framework.

A preliminary implementation of this framework is used to combine Inverse Entailment of the ILP system CProgol with a genetic search. Even though this implementation justifies the properness of the proposed framework, it could be improved in many ways. A natural improvement might be using more sophisticated genetic algorithms rather than using a simple genetic algorithm. For example the greedy cover set algorithm of CProgol, which repeatedly generalizes examples, could be replaced by a distributed genetic algorithm. The task-specific genetic operators can be used to guide the genetic search towards the interesting areas of the search space by specialization and/or generalization as it is done in usual concept learning systems. The fast evaluation mechanism can be used to compensate for the natural computation cost of a genetic algorithm and could lead to a high performance genetic search. In the current approach, the occurrence of atoms in a clause is not considered in the binary encoding of the clause and inactive atoms (e.g. unconnected predicates) are filtered from the induced hypotheses. This could lead to an incomplete search or inexact evaluation. Alternatively, the presence or absence of atoms in each clause can be encoded as a part of the binary representation of the clause. Finally, more experiments are required to evaluate the proposed framework in complex domains and noisy domains.

Undoubtedly, first-order concept learning and GAs are on opposite sides in the classification of learning processes (Kodratoff and Michalski, 1990). While GAs are known as empirical or BK-poor, first-order concept learning could be considered as BK-intensive method in this classification. We conclude that the framework proposed in this paper can be considered as a *bridge* between these two different paradigms to utilize the distinguishable benefits of each method in a hybrid system.

References

Anglano, C., Giordana, A., Bello, G. L., and Saitta, L. (1998). An experimental evaluation of coevolutive concept learning. In Shavlik, J., editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 19–27. Morgan Kaufmann.

Badea, L. and Stanciu, M. (1999). Refinement operators can be (weakly) perfect. In Džeroski, S. and Flach, P., editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 21–32. Springer-Verlag.

Bratko, I., Muggleton, S., and Varsek, A. (1991). Learning qualitative models of dynamic systems. In *Proceedings of the Eighth International Machine Learning Workshop*, San Mateo, Ca. Morgan-Kaufmann.

Feng, C. (1992). Inducing temporal fault dignostic rules from a qualitative model. In Muggleton, S., editor, *Inductive Logic Programming*. Academic Press, London.

Giordana, A. and Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation Journal*, 3(4):375–416.

Giordana, A. and Sale, C. (1992). Learning structured concepts using genetic algorithms. In Sleeman, D. and Edwards, P., editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 169–178. Morgan Kaufmann.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA.

Hekanaho, J. (1998). Dogma: A ga-based relational learner. In Page, D., editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, pages 205–214. Springer-Verlag.

Holland, J. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228.

Kennedy, C. J. and Giraud-Carrier, C. (1999). An evolutionary approach to concept learning with structured data. In *Proceedings of the fourth International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 1–6. Springer Verlag.

Kodratoff, Y. and Michalski, R. (1990). Research in machine learning: Recent progress, classification of methods and future directions. In Kodratoff, Y. and Michalski, R., editors, *Machine learning: an artificial intelligence approach*, volume 3, pages 3–30. Morgan Kaufman, San Mateo, CA.

Koza, J. R. (1991). *Genetic Programming.* MIT Press, Cambridge, MA.

Leung, K. S. and Wong, M. L. (1995). Genetic logic programming and applications. *IEEE Expert*, 10(5):68–76.

Michalski, R. (1980). Pattern recognition as rule-guided inductive inference. In *Proceedings of IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 349–361.

Mitchell, T. (1982). Generalisation as search. *Artificial Intelligence*, 18:203–226.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill, New York.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.

Muggleton, S. (1994). Inductive logic programming: derivations, successes and shortcomings. *SIGART Bulletin*, 5(1):5–11.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13:245–286.

Muggleton, S. (1999). Scientific knowledge discovery using Inductive Logic Programming. *Communications of the ACM*, 42(11):42–46.

Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo. Ohmsha.

Muggleton, S., King, R., and Sternberg, M. (1992). Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657.

Muggleton, S. and Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679.

Nienhuys-Cheng, S.-H. and de Wolf, R. (1997). *Foundations of Inductive Logic Programming.* Springer-Verlag, Berlin. LNAI 1228.

Plotkin, G. (1969). A note on inductive generalisation. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh.

Reiser, P. G. K. and Riddle, P. J. (1998). Evolving logic programs to classify chess-endgame positions. In Newton, C., editor, *Second Asia-Pacific Conference on Simulated Evolution and Learning*, Canberra, Australia.

Srinivasan, A., , Muggleton, R. K. S., and Sternberg, M. (1997). The predictive toxicology evaluation challenge. In *Proceedings of the Fifteenth International Joint Conference Artificial Intelligence (IJCAI-97)*, pages 1–6. Morgan-Kaufmann.

Varšek, A. (1993). *Inductive Logic Programming with Genetic Algorithms.* PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia.

Winston, P. H. (1970). *Learning Structural Descriptions from Examples.* Phd thesis, MIT, Cambridge, Massachusetts.

Wong, M. L. and Leung, K. S. (1997). Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5(2):143–180.

Exploratory Multi-Objective Evolutionary Algorithm: Performance Study and Comparisons

K. C. Tan†, E. F. Khor, C. M. Heng and T. H. Lee

Department of Electrical and Computer Engineering
National University of Singapore
10 Kent Ridge Crescent Singapore 119260
†Email: eletankc@nus.edu.sg

Abstract

An exploratory multi-objective evolutionary algorithm (EMOEA) has been proposed in previous publications. Its salient feature is to combine the properties of tabu search and evolutionary algorithm for effective multi-objective optimization. In addition, it also applies lateral interference, which is capable of distributing non-dominated individuals uniformly along the discovered Pareto-front at each generation without the need of any parameter setting. In this paper, the main objective is to perform extensive simulation studies to compare the performance of EMOEA against other evolutionary methods. Four benchmark test problems together with two well-known performance measures are applied. The studies have shown the competitive behavior of EMOEA to escape from local optima as well as to accurately identify the actual global optima in the noisy environment.

1 INTRODUCTION

Evolutionary algorithms have been recognized to be well suited for MO optimization problems (Fonseca and Fleming, 1993). Unlike conventional methods that linearly combine multiple attributes to form a composite scalar objective function, evolutionary algorithm for MO optimization incorporates the concept of Pareto's optimality or modified selection schemes to evolve a family of solutions at multiple points along the trade-off surface simultaneously. Since Schaffer's work (1985), evolutionary techniques for MO optimization have been gaining significant attentions from researchers in various fields, which are reflected by the high volume of publications in this topic in the last few years (over 25 Ph.D. theses, more than 80 journal papers, and more than 300 conference papers). For more information on various techniques of handling multi-objective optimization problems via evolutionary algorithms, readers may refer to the literatures of (Coello Coello, 1998; Van Veldhuizen and Lamont, 1998; Zitzler and Thiele, 1998).

(Tan *et al*., 2001) has proposed an exploratory multi-objective evolutionary algorithm (EMOEA), which incorporates the memory-based feature of tabu search (TS) to maintain the stability of MO optimization towards a global and uniform Pareto-front. The hybridization of TS in evolutionary optimization helps to improve the MO search performances by avoiding repeats of currently found peaks, i.e., local optima in the search space is avoided while good regions are being well explored. Besides, a method of lateral interference, which is highly efficient of distributing non-dominated individuals uniformly along the discovered Pareto-front was also proposed in (Tan *et al*., 2001). It can be performed without the need of any parameter setting and can be flexibly applied in either parameter or objective domain depending on the nature of the optimization problem involved. Seeing the needs to further examine and evaluating the method, this paper is produced to do the performance comparison studies of EMOEA against other well-known evolutionary methods quantitatively. Four test problems that exhibit important characteristics, suitable for validating the effectiveness and efficiency of MO optimization methods in converging to the Pareto-optimal front and maintaining population diversity in the current non-dominated front, have been applied in the studies. This paper is organized as follow: The elementary concepts of EMOEA are introduced in Section 2, performance comparisons are performed in Section 3 while conclusions are drawn in Section 4.

2 EXPLORATORY MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM

In general, multi-objective (MO) optimization can be defined as the problem of optimizing a vector of non-commensurable and often competing objectives or cost functions, viz, it tends to find a parameter set $\boldsymbol{P}$ for

$$\underset{\boldsymbol{P}\in F}{\text{Min}}\ \boldsymbol{F}(\boldsymbol{P}),\ \boldsymbol{P}\in \boldsymbol{R}^n \qquad (1)$$

where $\boldsymbol{P} = \{p_1, p_2, \ldots, p_n\}$ is a individual vector with n parameters and $\boldsymbol{F}$ defines a set of individual vectors. $\{f_1, f_2, \ldots, f_m\}$ are m objectives to be minimized and $\boldsymbol{F} = \{f_1, f_2, \ldots, f_m\}$. Instead of a single optima, solution to MO optimization problem is often a family of points known as Pareto optimal set, where each objective component of any point along the Pareto-front can only be improved by degrading at least one of its other objective components (Richardson *et al*., 1989; Srinivas and Deb, 1994). In the total absence of information regarding the preferences of objectives, ranking scheme based upon the Pareto optimality is regarded as an appropriate approach to represent the strength of each individual in an evolutionary algorithm for MO optimization (Fonseca and Fleming 1993; Srinivas and Deb, 1994). A vector $\boldsymbol{F}_a$ is said to dominate another vector $\boldsymbol{F}_b$, denoted as $\boldsymbol{F}_a \prec \boldsymbol{F}_b$, *iff*

$$f_{a,i} \le f_{b,i}\ \forall i \in \{1,2,\ldots,m\}$$
$$\text{and } \exists\ j \in \{1,2,\ldots,m\} \text{ where } f_{a,j} < f_{b,j} \qquad (2)$$

The Pareto ranking scheme assigns the same smallest cost for all non-dominated individuals, while the dominated individuals are ranked according to how many individuals in the population dominating them. So, the rank of an individual x in a population can be given by $rank(x) = 1 + q_x$, where q_x is the number of individuals dominating the individual x in the objective domain (Fonseca and Fleming, 1993). They also extended the Pareto's domination scheme in their proposed multi-objective genetic algorithm (MOGA) to include goal and priority information for MO optimization. Although MOGA is a good approach, the algorithm only allows a single goal and priority vector setting, which may be difficult to define in *a-priori* to an optimization process (Coello Coello, 1998).

With a modified Pareto-domination scheme, Tan *et al*. (1999) proposed a unified multi-objective evolutionary algorithm (MOEA) that is capable of comparing the domination among individuals for multi-objective optimization dealing with both soft and hard optimization constraints. The scheme also allows the incorporation of multiple goals and priorities with different combinations of logical "AND" and "OR" operations for greater flexibility and higher-level decision support. Extending from the Pareto's domination and ranking schemes of Tan *et al*., (1999), (Tan *et al*., 2001) has proposed an exploratory multi-objective evolutionary algorithm (EMOEA) that incorporates the lateral interference and memory-based feature of tabu search to maintain the stability of MO optimization towards a global and uniform Pareto-front. Sections 2.1 briefly explains the working principle of lateral interference while Section 2.2 describes the overall algorithm of EMOEA including tabu-based individual examination rule.

2.1 LATERAL INTERFERENCE (LI)

Among the methods to evolve an equally distributed population along the Pareto-front and to distribute the population at multiple optima in the search space, the 'niche induction' technique by means of a sharing function (Deb and Goldberg, 1989) is the most popular approach. This method creates sub-divisions in the objective domain by degrading an individual's fitness upon the existence of other individuals in its neighborhood defined by a shared distance.

To avoid the setting of shared distance, a population distribution method is proposed in (Tan *et al*, 2001). It is capable of uniformly distributing all individuals along the Pareto-front for MO optimization without the need of any parameter setting. It can be applied in either the parameter domain or in the objective domain as needed. The method is called *Lateral Interference* which and it works based upon the principles of exploitation competition and interference competition (Encyclopaedia Britanica, 2000), which form the basis of distributing population uniformly along the Pareto-front in MO optimization without the need of any parameter setting. According to the first principle (exploitation competition), individuals with higher fitness or lower cost values will always be more likely to win when compete with individuals with lower fitness or higher cost values. The second principle (interference competition) only takes place among the individuals with same level of fitness or cost, or in other words, individuals that are equally strong in the exploitation competition.

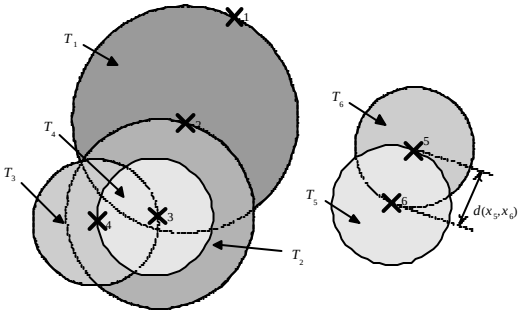


Fig. 1 Illustrative Example for Interference Competition

Fig. 1 illustrates the determination of territory in lateral interference for each individual among individuals x_1-x_6, as depicted in cross-mark, in the concerned two dimensional space which may be either in the parameter domain or the objective domain. All of the individuals in this figure are assumed to be at the same level of fitness or Pareto rank values so that the inference competition takes place among them for limiting resources.

Table 1 Illustrative Example for Computation of Severity or Impact of Being Interfered with

Inhibited individual	Territories of other individual(s) that it belongs to	$H_s(j)$
x_1	None	0
x_2	T_1	1
x_3	T_1, T_2, and T_4	3
x_4	T_2, and T_3	2
x_5	T_6	1
x_6	T_5	1

The resulted territories are labeled as T_i for individual i, $\forall$ i = 1, 2,..., 6, which are represented as shaded circles in different intensity for better visualization. As can be seen, the territory for individual that is relatively far away, e.g., individual x_1, from other individuals is assigned a larger territory than those that are closed together, e.g. individuals x_3 and x_4. Based on the territory of each individual, the impact of being interfered with, denoted as $H_s(j)$ for individual j, $\forall$ j = 1, 2,..., 6, can be counted according to how many territories of other individuals that the individual j lies at or belongs to, as shown in Table 1.

Before the lateral interference, the non-dominated individuals in the population are classified into one category, with similar dummy cost value. To maintain the diversity of the population, these classified individuals are undergone lateral interference and the resulted severity (H_s) of interference for each classified individual is added to its dummy cost. This group of individuals is then ignored and another layer of non-dominated individuals (ignoring the previously classified ones) is considered, where the assigned dummy cost is set higher than the highest interference cost of the previously classified individuals. This process continues until all individuals in the individual list have been classified. The final resulted individuals' dummy cost value after the lateral interference is referred here as interference cost. The smaller value of the interference cost, the better is the individual.

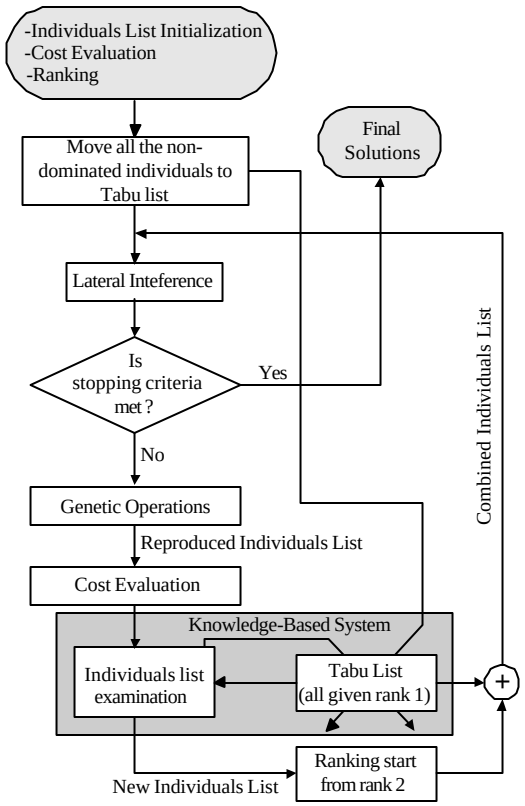


Fig. 2 Overall Program Flowchart of EMOEA

2.2 FLOW CHART OF EMOEA

The overall program flowchart of the EMOEA algorithm (Tan *et al*., 2001) is shown in Fig. 2. At the initial stage of evolution, a list of $N_c^{(0)}$ number of individuals is initialized, where $N_c^{(0)}$ is the size of individual list in the evolution. The individual list is then decoded to parameter vectors for cost evaluation. Subsequently, all the evaluated individuals are ranked according to the specifications assigned. All the non-dominated individuals are copied to the empty tabu list while the rest of the individual list are fed to lateral interference to compute for interfered cost as described in Section 2.1. If the stopping criterion is not met, genetic operations will be applied to the evaluated individuals. Here, simple genetic operations consist of tournament selection based on interfered cost, standard crossover and mutation are performed to reproduce offspring for the next generation.

After that, the resulted reproduced individual list with size N_c will be evaluated and examined by the tabu-based individual examination scheme. This scheme is equipped with knowledge of and tabu list to further enhance the search performance by avoiding repeats of previous exploration path to the found peaks. Starting from the first individual in the reproduced individual list, if the examined individual dominate any member in the tabu list, the individual will replace the dominated member(s) in the tabu list. Otherwise, if the individual is dominated by any member in the tabu list, it will be kept in the individual list if the individual is not a tabu. If both the conditions are not satisfied, the individual will be rejected from the individual list and prohibited from surviving in the next generation. In case that the individual is not dominated by any member in the tabu list and if the tabu list is not full i.e., its size does not achieve the maximum limit, the individual will be added to the tabu list. Otherwise, if the individual is able to interfere with more

than one member in the tabu list within its territory, it will be allowed to replace the tabu member that has the shortest distance from the individual in the space concerned. This is to promote the uniform distribution of the tabu list. If the condition is not met, the individual will be kept in the individual list if any of its objective components is better than the best objective component value found in the tabu list or the individual is not a tabu. Besides encouraging long distance exploration to provide better exploration for other possible peaks in the solution space, it is also capable of maintaining stability of evolution towards the global and uniform Pareto-front.

Subsequently the resulted new individual list as well as the updated tabu list are fed into the MO genetic evolution to form the combined individual list, which has the size of $N_c = N_c + N_t$, where N_t is the size of tabu list. This combined individual list is fed to the next generation of evolution and this process is repeated until the stopping criteria is met.

3 PERFORMANCE COMPARISONS

3.1 THE TEST PROBLEMS

Table 2 summarizes the test problems (T.P.). These test problems are considered as they include some important characteristics that are suitable for validating the effectiveness and efficiency of MO optimization methods in converging to the Pareto-optimal front and maintaining population diversity in the current non-dominated front.

Table 2 Features of Test Problems

T.P.	Features
1	Non-convex Pareto-optimal front
2	Discontinuous Pareto optima in search domain
3	Multi-modal and deceptive problem with harmful local peak
4	Noisy and landscape

3.1.1 Test problem 1

This is the Fonseca' s two-objective minimization problem which has been widely studied by (Fonseca and Fleming, 1993). Besides its non-convex Pareto optimal front, this test function has large and non-linear trade-off curve that should challenge the MO evolutionary algorithm' s ability to find and maintain the entire front uniformly. Besides, it is easy for visualization and comparison. The two-objective functions, f_1 and f_2, to be minimized are given as

$$f_1(x_1,..., x_8) = 1 - exp\left(- \sum_{i=1}^{8} \left(x_i - \frac{1}{\sqrt{8}} \right)^2 \right) \quad (3a)$$

$$f_2(x_1,..., x_8) = 1 - exp\left(- \sum_{i=1}^{8} \left(x_i + \frac{1}{\sqrt{8}} \right)^2 \right) \quad (3b)$$

where $-2 \le x_i < 2, \ \forall i = 1,2,...,8$. The trade-off line is shown by the curve in Fig. 3, where the shaded region represents the unfeasible area in the objective domain.

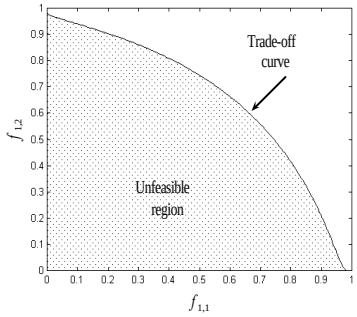


Fig. 3 Trade-off Curve in the Objective Domain for Test Problem 1

3.1.2 Test problem 2

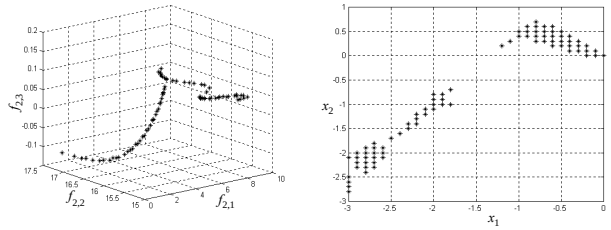
This problem focuses on the feature of discontinuity of Pareto optima in search domain. It uses the simultaneous minimization of three objective functions f_1, f_2 and f_3 depending on two real-valued parameters, x_1 and x_2 (Viennet *et al*., 1996):

$$f_1 = 0.5\left(x_1^2 + x_2^2\right) + \sin\left(x_1^2 + x_2^2\right), \quad (4a)$$

$$f_2 = \frac{(3x_1 - 2x_2 + 4)^2}{8} + \frac{(x_1 - x_2 + 1)^2}{27} + 15, \quad (4b)$$

$$f3 = \frac{1}{\left(x_1^2 + x_2^2 + 1\right)} - 1.1 \times \exp\left(-x_1^2 - x_2^2\right) \quad (4c)$$

where $-3 \le x_i \le 3, \ \forall i = 1,2$. This problem has Pareto-front appears to be a three dimensional curve as shown in Fig. 4a. It is able to challenge the MO optimization approaches (Van Veldhuizen and Lamont, 1999). Moreover, f_1 and f_3 respectively, present four and two local minima. The Pareto-optimum points are shown in Fig. 4b where they are discontinuous in the parameter domain. These can be explained by two facts. Firstly the three optimal points in each individual objective function f_1, f_2 and f_3 are Pareto-optimal of the overall three objective functions, and secondly f_1 and f_3 have local minima.



(a) Objective Domain (b) Parameter Domain

Fig. 4 Pareto Optimal Set of Test Problem 2

3.1.3 Test Problem 3

It is originated from Deb (1999) for two-objective minimization with the existing of local optimum where search algorithms are easily to be trapped. In this paper, the original test problem is modified and expanded such that the global optimum is farther away from the local optimum and the dimensionality of the search space is higher. The purpose of these modifications is to achieve higher level of optimization difficulties in the sense that it provides more tendencies for the search algorithms to pre-maturely converge to local optimum, and has less possibility to discover the global optimum. The modified two-objective functions to be minimized are:

$$f_1 = x_1, \tag{5a}$$

$$f_2 = \frac{1}{x_1}\prod_{i=1}^{3} g_i, \tag{5b}$$

where, $$g_i = 2.0 - \exp\left\{-\left(\frac{x_{i+1}-0.1}{0.004}\right)^2\right\} - 0.8\exp\left\{-\left(\frac{x_{i+1}-0.9}{0.4}\right)^2\right\}, \quad \forall i = 1,2,3 \tag{5c}$$

where, $$0.1 \le x_1 \le 1, \tag{5d}$$

and, $$0 \le x_j \le 1, \quad \forall j = 2,3,4 \tag{5e}$$

Fig. 5 depicts the function of g_i for $0 \le x_{i+1} \le 1$. As can be seen, g_i is a bimodal function with x_{i+1} = 0.1 as global minimum and x_{i+1} = 0.9 as local minimum, where the distance (0.8) in between has been increased by 100% from the original problem with a distance of 0.4 (Deb, 1999). Fig. 6 shows the f_1-f_2 plot, with local and global Pareto-optimal curve represented by dashed and solid line, respectively. The shaded region represents the unfeasible area.

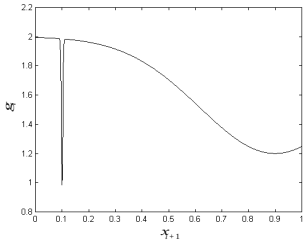


Fig. 5 g_i Has a Global and a Local Minima

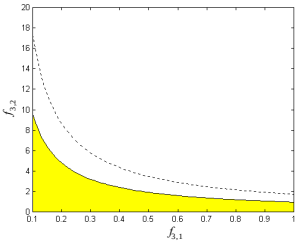


Fig. 6 Global and Local Pareto-Optimal Curve

3.1.4 Test Problem 4

On this test problem, the search algorithms are evaluated in noisy environment to test their robustness in the sense that the disappearance of important individuals from the population has little effect on the global evolution behavior (Collard and Escazut, 1995). In order to investigate the relative abilities of MO search algorithms in noisy environment, noisy version of two-objective optimization with three variables is constructed here the function being optimized contains the elements of noise:

$$f_1 = x_1', \tag{6a}$$

$$f_2 = \frac{1}{x_1}\{1+g\}, \tag{6b}$$

$$g = \left(x_2'^2 + x_3'^2\right)^{0.25}\left[\sin^2\left(50\left(x_2'^2 + x_3'^2\right)^{0.1}\right)+1.0\right] \tag{6c}$$

Instead of performing the optimization on the 'real' parameters, x_i, the optimization is performed on the 'corrupted' parameters with additive noise elements:

$$x_i' = x_i + N(\mu, \sigma), \tag{6d}$$

where $0.1 \le x_1 \le 1$; $-100 \le x_i \le 100$ $\forall i = 2,3$ and $N(\mu,\sigma)$ is a white noise. Note that the noisy search environment is modeled with the corrupted parameters. This is to provide noisy global optimum points in parameter domain while maintaining the global-front in objective domain for easier comparison and illustration.

Besides the noisy environment, the optimization difficulty is further enhanced by multi-model with different patterns of their well depths and heights of the barriers between wells as formulated in eqn. 6c originated from Schaffer *et al.*, (1989). The 2-dimensional cross section of $f_2(x)$-x_1 through the origin is shown in Fig. 7 and it can be seen that there exists plenty of local optima around the global optima. The Pareto-optimal curve in the objective domain is shown in Fig. 8 and the shaded region represents the unfeasible area.

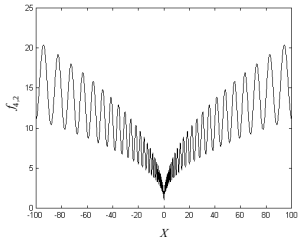


Fig. 7 Central Cross Section of f_2

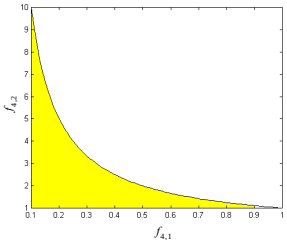


Fig. 8 Pareto-Optimal Curve in Objective Domain

3.2 SIMULATION RESULTS

Beside EMOEA, other MO evolutionary optimization methods concerned in this comparison study include VEGA from (Scaffer, 1985), NPGA from (Horn and Nafpliotis, 1993), MOGA from (Fonseca and Fleming, 1993), NSGA from (Srinivas and Deb, 1994), SPEA from (Zitzler and Thiele, 1999) and MOEA from (Tan *et al.*, 1999). These methods have often been applied or taken as

reference in literature for comparing different population-based multi-objective evolutionary algorithms.

Phenotype sharing is applied in all algorithms that apply the sharing operation. A standard shared distance value of 0.01 in the normalized space are set for MOGA, NSGA, and NPGA as well as for the performance measure of *UD*. MOEA applies the adaptive sharing scheme to determine a suitable sharing distance at every generation (Tan *et al*., 1999), while SPEA and EMOEA does not require any sharing distance parameters. Tournament selection is applied in MOGA, SPEA, MOEA, and EMOEA with a tournament size of 2 as they usually applied in their respective original articles. For the Pareto tournament selection in NPGA, t_{dom} = 10% of the population size is used since it has been recommended by (Horn *et al*., 1994) as an optimum value for tight and complete distribution. Note that all algorithms considered are implemented with same coding scheme, crossover and mutation operations. Each parameter is represented by 3-digit decimal code and concatenated to form the chromosomes which results in a shorter chromosome length and avoids the *Hamming-cliff* effect as encountered in binary-based coding scheme (Tan *et al*., 1999). Also, standard mutation with probability of 0.01 and standard two-point crossover with probability of 0.7 are used in all cases.

All methods under the comparison study are implemented with the same common sub-functions using the same programming language in Matlab (The Math Works, 1998) on an Intel Pentium II 450 MHz computer. Each of the simulation is terminated automatically when a pre-specified simulation period for each test problem is reached, in the same platform that is free from other computation or being interrupted by other program. The period for all algorithms being compared for test problem 1, 2, 3 and 4 are set as 180, 60, 100 and 100 sec., respectively. These periods are pre-determined based on the criteria that they are found most appropriate to clearly observe the difference of simulation results among the various methods, in which at least one method has converged satisfactorily. 30 independent simulation runs are performed for each method on each test problem so as to study the statistical performance such as consistency and robustness of the methods. Note that a random initial population is created for each of the 30 runs, and for each test problem, all methods are operated on the same 30 independent initial populations.

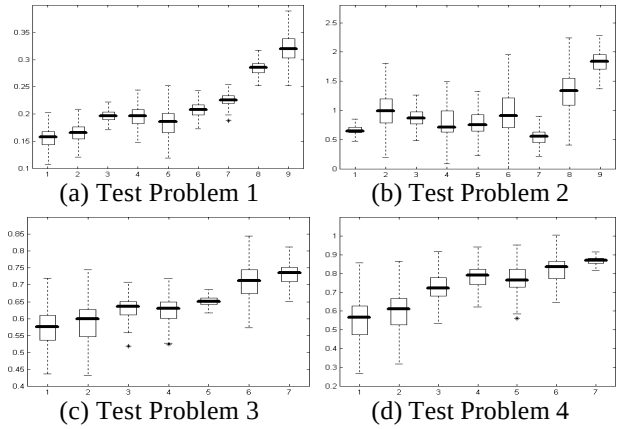
As according to (Zitzler and Thiele, 1999), three combinations of population $\{P, P'\}$ = {95, 5}, {70, 30} and {30, 70}, where $P + P'$ = 100 are used on test problem 1 and 2 for SPEA. For test problem 3 and 4, only one optimum combination of population {30, 10} is used for SPEA. The combination of $\{N_c, N_t\}$ in EMOEA for test problem 1-4 are {100, 20}, {100, 80}, {30, 10} and {30, 10}, respectively. For all other methods, a population size of 100 and 30 is applied for test problems 1-2 and 3-4, respectively. Note that all results are evaluated upon the final population of each simulation run. For SPEA and EMOEA that consists of two sub-populations, its combined population of $P + P'$ at the final generation is concerned. The indexes for the different algorithms on each test problem are listed in Table 3, where SPEA 1, 2, and 3 representing the different combination of $\{P, P'\}$ as mentioned above.

Fig. 9 summarizes the simulation results for each algorithm on each test problem with respect to performance measures of size of space covered SSC (Zitzler and Thiele, 1999). The distribution simulation data of 30 independent runs is represented in the box plot format (Chambers *et al*., 1983) to visualize the distribution of simulation data. In each graph, the sequence of box plots from left to right is based on the above indexes of algorithms. Concerning the measure of size of space cover (*SSC*), the performance of EMOEA is most outstanding in discovering a large Pareto-front as compared with other algorithms, especially on the test problems 1 and 2. This is followed by MOEA and SPEA as evidence on all test problems under studied. For the rest of algorithms, no significant difference of *SSC* is observed.

The performance measures of $C(X_i, X_j)$ for the comparison sets between algorithms i and j where, i, j = 1, 2,…, 9 on test problems 1-2 as well as i, j = 1, 2,…, 7 on test problems 3-4, are shown in Fig. 10. Again box plots are used to summarize the sample distributions of 30 independent runs per each case. In each rectangle containing box plots, the sequence of box plots from left to right is based on the indexes of algorithm as listed in Table 3. The ranges of y-axis and x-axis of each graph are [0, 1] and [1, number of compared algorithms on the respective test problem. As can be seen, $C(X_i, X_j)$ for $i = j$ always take the value of zero since the two identical populations cannot dominate each other.

Table 3 Algorithm Indexes for Each Test Problem

Test Problems	1	2	3	4	5	6	7	8	9
1	VEGA	NPGA	MOGA	NSGA	SPEA1	SPEA2	SPEA3	MOEA	EMOEA
2	VEGA	NPGA	MOGA	NSGA	SPEA1	SPEA2	SPEA3	MOEA	EMOEA
3	VEGA	NPGA	MOGA	NSGA	SPEA	MOEA	EMOEA	-	-
4	VEGA	NPGA	MOGA	NSGA	SPEA	MOEA	EMOEA	-	-



(a) Test Problem 1 (b) Test Problem 2

(c) Test Problem 3 (d) Test Problem 4

Fig. 9 Box Plots for the Measures of *SSC*

For example, the rectangle $C(X_1, X_{1\text{-}7})$ on test problem 3 indicates that solutions of VEGA (given by X_1) almost fails to dominate any solutions of other algorithms (given by $X_{2\text{-}7}$) while the rectangle $C(X_7, X_{1\text{-}7})$ on the same problem implies that solutions of EMOEA (given by X_7) dominate almost all of the solutions of ($X_{1\text{-}6}$). In general, the overall results show that, except the $C(X_3,X_{1\text{-}9})$ on test problem 2 where SPEA and MOEA are slightly superior than EMOEA, EMOEA provides the best or equally best non-dominated individuals compared to other approaches. Besides, other minor observations can also be made on the basis of each test problem. For test problems 3 and 4, it is noticeable (concerning the rectangles $C(X_7, X_{1\text{-}7})$ on both test problems 3 and 4) that EMOEA dominates other algorithms most obviously as compared to other algorithms in the respective test problems. Also, EMOEA is dominated the least by any other algorithms on test problems 3 and 4, except rectangle $C(X_4, X_{1\text{-}7})$ where MOEA is dominated the least by NSGA.

From our experiment, a few findings can be observed. Although VEGA, NPGA, MOGA and NSGA require less computational effort, their final populations are shown to be worse than SPEA, MOEA and EMOEA in terms of the performance measures of *SSC* and *C*. This may be caused by the absence of preserved strategy or other genetic operations that are different from SPEA, MOEA and EMOEA. For SPEA, although it applies the preservation strategy by means of external population and clustering, the produced final populations are not the best among the algorithms under studied. This may due to the large computational effort required to realize the algorithm such as the clustering operation, which limits the number of iterations it may perform in a given fixed period of CPU time. Although the needed computational effort for EMOEA is moderate among all algorithms and is less than SPEA, it perform well in relative to all other algorithms especially in the measures of *SSC* and *UD*. This is particularly evident on test problems 3 and 4, where EMOEA is shown to have higher capability to reach the global Pareto-front and escape from the harmful local optima on test problem 3, and has shown to be more robust to external noise than others for searching in noisy environment which enables it to track the actual Pareto-front effectively and accurately.

4 CONCLUSIONS

Extensive validation and comparison of EMOEA and other famous MO optimization methods have also been performed upon four benchmark problems, and their performances have been compared quantitatively in this paper. Simulation results unveiled that the overall performance of EMOEA is relatively well in searching and maintaining for the uniformly distributed non-dominated solutions along the global Pareto-front as reported. Besides, the studies also shows that EMOEA has a competitive behavior to escape from local optima as well as to accurately identify the actual global optima in the noisy environment when compared against recent methods. Nevertheless, the role of EMOEA in dealing with other types of test problems (i.e. such as heavy function evaluation, heavy constraint, dynamic search space etc.) requires further investigations

References

Chambers, J. M., Cleveland, W. S., Kleiner, B., and Turkey, P. A. (1983). *Graphical methods for data analysis*. Wadsworth & Brooks/Cole, Pacific CA.

Coello Coello, C. A. (1998). *An Updated Survey of GA-based Multiobjective Optimization Techniques*. Technical Report: Lania-RD-98-08, Laboratorio Nacional de Informatica Avanzada (LANIA), Xalapa, Veracruz, Mexico.

Collard, P., and Escazut, C. (1995). Genetic operators in a dual genetic algorithm. *International Conference on Tools and Artificial* Intelligence, pp. 12-19.

Deb, K. (1999). Multi-objective genetic algorithms: Problem difficulties and construction of test problem. *Journal of Evolutionary Computation*, The MIT Press, vol. 7(3), pp 205-230.

Deb, K., and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In Schaffer, J. D. ed., *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 42-50.

Encyclopaedia Britannica. (2000). The Encyclopedia Britannica. http://www.britannica. com/ bcom/ eb/ article /2/0,5716,127612+29,00.html.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithm for multiobjective optimization, formulation, discussion and generalization. In *Forrest*, S., ed. *Genetic Algorithms: Proceeding of the Fifth International Conference.* Morgan Kaufmann, San Mateo, CA, pp. 416-423.

Horn, J. and Nafpliotis, N. (1993). *Multiobjective optimization using the niche Pareto genetic algorithm.*

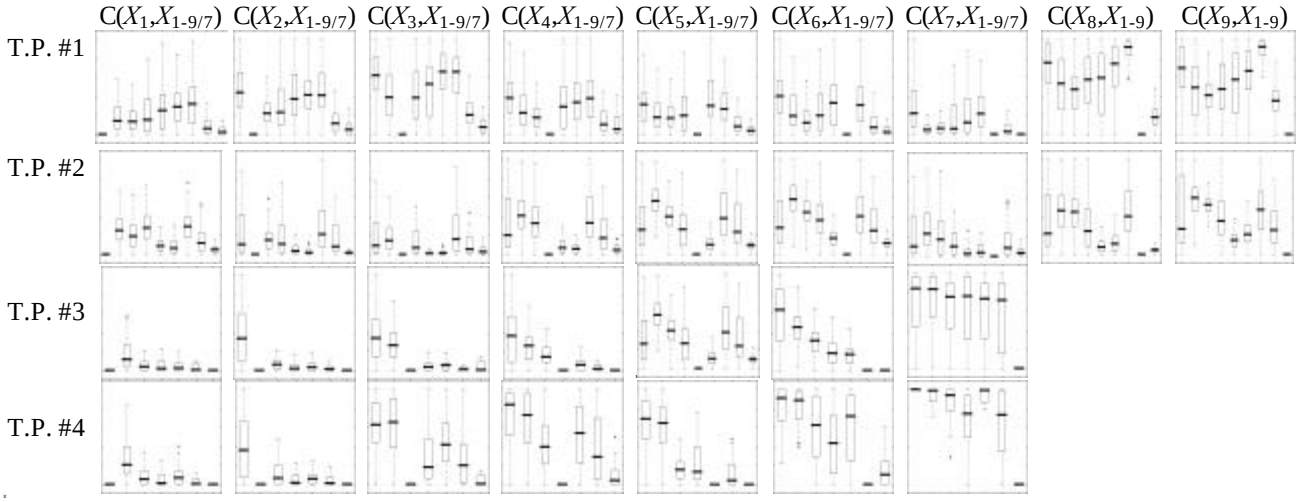


Fig. 10 Box Plot for *C* Measure. Each rectangle, refers to the measure of $C(X_i, X_{1\text{-}n})$, represented by box plots arranged left to right, between *i* algorithm and algorithms ranging from 1 to *n* where *n* is 9 for test problems 1-2 while 7 for test problems 3-4.

IlliGAL Report 93005, University of Illinois at Urbana-Champain, Urbana, Illinois, USA.

Horn, J., Nafpliotis, N. and Goldberg, D.E. (1994). A niched Pareto genetic algorithm for multiobjective optimisation. *Proceeding of First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence,* vol. 1, pp. 82-87.

Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer, ed., *Proc. of Third Int. Conf. on Genetic Algorithms*, pp. 191-197.

Schaffer, J. D. (1985). Multiple-objective optimization using genetic algorithm. *Proceedings of the First International Conference on Genetic Algorithms*, pp. 93-100.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proceedings of Third International Conference on Genetic Algorithms*, pp. 51-60.

Srinivas, N. and Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, MIT Press Journals, vol. 2, no. 3, pp. 221-248.

Tan, K. C., Lee, T. H. and Khor, E. F. (1999). Evolutionary algorithms with goal and priority information for multi-objective optimization. *IEEE Proceedings of the Congress on Evolutionary Computation*, Washington, D.C, USA, vol. 1, pp. 106-113.

Tan, K. C., Lee, T. H., and Khor, E. F. (2001). Tabu-based exploratory evolutionary algorithm for effective multi-objective optimization. *Accepted by First International Conference on Evolutionary Multi-Criteria Optimization (EMO' 01).*

The Math Works, Inc. (1998). *Using MATLAB*, The Math Works Inc., version 5.

Van Veldhuizen, D. A., and Lamont G. B. (1999). Multiobjective evolutionary algorithm test suites. *Symposium on Applied Computing*, San Antonio, Texas, pp. 351-357.

Van Veldhuizen, D. A., and Lamont, G. B. (1998). Evolutionary computation and convergence to a Pareto front. In Koza, J. R., ed., *Late Breaking Paper at the Genetic Programming 1998 Conference*, Stanford University Bookstore, Stanford University, California, pp. 221-228.

Viennet, R., Fonteix, C., and Marc, I. (1996). Multicriteria optimization using a genetic algorithm for determining a Pareto set. *International Journal of Systems Science*, vol. 27, no. 2, pp. 255-260.

Zitzler, E., and Thiele, L. (1998). *An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach*. Technical Report 43, Computer Engineering and Communication Network Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

Zitzler, E., and Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp.257-271.

Nonlinear Constraint Handling Technique via Angular Transformation

K. C. Tan†, T. H. Lee, E. F. Khor, C. M. Heng and D. Khoo

Department of Electrical and Computer Engineering
National University of Singapore
10 Kent Ridge Crescent Singapore 119260
†Email: eletankc@nus.edu.sg

Abstract

This paper proposes an angular transformation methodology that transforms the non-linear optimization constraints into the gene domains in evolutionary algorithms and as such, trimming away sections of unfeasible regions in constraint optimization problems. This results in a smaller search space and reduces the efforts of evolution in finding the global optimum solution. In addition, the proposed method can be incorporated in many objective domain based methods to remove some of the unfeasible regions before applying these methods and are compatible with standard genetic operators like crossover and mutation without the need of rejecting/repairing any unfeasible solutions as adopted in most existing methods.

1 INTRODUCTION

Practical optimization problems often involve optimizing a set of objective components within a pre-specified feasible region constrained in parameter domain. As stressed by (Hoffmeister and Sprave, 1996), they often not only have a high complexity with respect to the number of decision variables or parameters but also have a possibly large number of constraints to be satisfied for feasible solution. Recently, many researchers have proposed various constraint handling approaches for evolutionary computations (ECs). These includes methods based on:

- preserving feasible solutions (Schoenauer and Michalewicz, 1996),
- penalty functions (Homaifar *et al*., 1994; Joines and Houck, 1994; Michalewicz and Attia, 1994),
- search for feasible solutions (Schoenauer and Xanthakis, 1993).

Although the above methods have shown some successful results, they share the potential to produce new chromosome strings that may or may not feasible (Koziel and Michalewicz, 1999). This leads to additional computation time to evolve and evaluate unfeasible offspring that later need to be penalized or eliminated through special operators. In addition, in problems where feasible solutions are difficult to find, the whole population may suddenly converge on a feasible string when it is found, despite the fact that this feasible string may be far away from the optima. To address the issue, an interesting work, reported from Koziel and Michalewicz (1999), has recently proposed an alternative approach to the constraint handling which is based on homomorphous mapping T between the n-dimensional cube $[-1, 1]^n$ and the feasible part of the search space to guarantee a feasible solution. Nevertheless, beside other drawbacks as mentioned in (Koziel and Michalewicz, 1999), this method introduced a mapping function T needed to determine experimentally before the run of the algorithm.

Regarding the above consideration, this paper proposes an alternative method of angular transformation, for use with ECs or any other algorithms that represent solution genetically, to handle non-linear constraint optimization problem. The gene search space produced is the feasible region and because of this, the solution is always feasible and no special crossover or mutation operators are necessary to evolve the population. This approach, later explained in this paper, is a very promising direction of research in evolutionary optimization.

2 ANGULAR TRANSFORMATION

Angular transformation is a gene domain constraint handling technique that is capable of handling nonlinear equality/inequality constraints. Although the angular transformation proposed in this section may not be able to handle all types of constraints, it can be effectively used to trim away part of the unfeasible regions such that the EA has a smaller area to search for the global optimum.

2.1 BASIC MECHANISM

The basic principle is to evolve the ratio of each parameter involved. After the ratio is obtained, the actual parameter value is derived from the constraint. Each parameter has a constant value, which will be known as R,

multiplied with its ratio to produce the final value for that parameter.

In order to help explain angular transformation, let there be an optimization problem that has an equality constraint:

$$f(p_1, p_2, \ldots p_n) = C \quad (1)$$

where p_i is parameter number i from a set of n parameters. For each parameter p_i, let:

$$p_i = Rx_i \quad (2)$$

R is the constant described earlier to be multiplied with the ratio x_i of parameter i. By substituting eqn. 2 into eqn. 1, eqn. 3 is produced:

$$f(Rx_1, Rx_2, \ldots Rx_n) = C \quad (3)$$

If the EA is used to evolve a set of values for x_i, and C is user-defined, the only unknown variable in eqn. 3 is R. By entering the set of x_i values and C into eqn. 3, R can be derived. Now, if R and x_i substitutes their counterparts in eqn. 2 to obtain p_i, the result is a set of parameters p_i that fulfills the constraint in eqn. 1. This is the basic mechanism of angular transformation. Note that angular transformation is not limited to handling equality constraints. An inequality constraint can be converted into the form of constraint presented in eqn. 1 by adding a dummy parameter. A generic double-sided inequality constraint looks like eqn. 4:

$$C_L \le f(p_1, p_2, \ldots p_n) \le C_H \quad (4)$$

By adding a dummy parameter d, the constraint becomes:

$$\begin{aligned} f(p_1, p_2, \ldots p_n) + d &= C_H \\ 0 \le d &\le C_H - C_L \end{aligned} \quad (5)$$

A single-sided inequality with a lower limit looks like eqn. 6a:

$$C_L \le f(p_1, p_2, \ldots p_n) \quad (6a)$$

Eqn. 6a can be converted into the equality constraint eqn. 6b or eqn. 6c:

$$\begin{aligned} f(p_1, p_2, \ldots p_n) + d &= C_L \\ d &\le 0 \end{aligned} \quad (6b)$$

$$\begin{aligned} f(p_1, p_2, \ldots p_n) &= C_L + d \\ d &\ge 0 \end{aligned} \quad (6c)$$

A generic single-sided inequality constraint with only an upper limit looks like eqn. 7a:

$$f(p_1, p_2, \ldots p_n) \le C_H \quad (7a)$$

Eqn. 7a can be converted into the equality constraint shown in eqn. 7b:

$$\begin{aligned} f(p_1, p_2, \ldots p_n) + d &= C_H \\ d &\ge 0 \end{aligned} \quad (7b)$$

The EA only evolves the set of x_i while R is allowed to adapt freely to the constraints. As such, any genetic boundary on x_i does not necessarily result in a fixed boundary on p_i.

2.2 THE ANGULAR CONSTRAINT

This section describes an early attempt to restrict the evolution of the set of p_i values. First, consider a multi-dimensional search space where each axis corresponds to a single parameter. Within this search space, the equality constraint can be represented as a hyperplane. A straight line is drawn from the origin to meet the constraint hyperplane. Using the length from the origin to the point of intersection and the angle between this line and each axis, the set of parameters that satisfy the constraints can be derived. First, a two objective constraint $f(p_1, p_2) = C$ is considered. This constraint is illustrated on the two-dimensional search space in Fig 1. A line is drawn from the origin O to meet the constraint. By varying the angle θ_1, the gradient of this line is determined. This line intersects with the constraint line at point A. The length of OA is R and the angle between OA and each axis are θ_1 and θ_2 respectively.

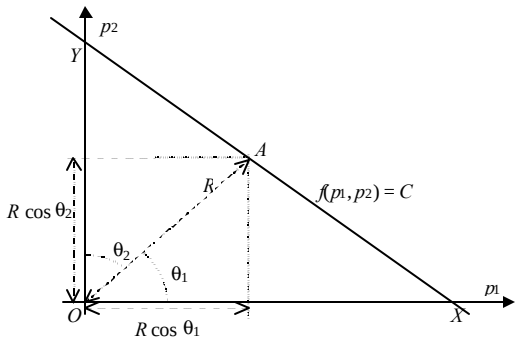


Fig. 1 Two Parameter Example

From Fig 1, it can be seen that:

$$p_1 = R \cos \theta_1 \quad (8a)$$

$$p_2 = R \cos \theta_2 \quad (8b)$$

Using Pythagoras' Theorem to derive the relation between the $R \cos \theta_i$ of each parameter:

$$p_1^2 + p_2^2 = R^2$$

$$R^2 \cos^2\theta_1 + R^2 \cos^2\theta_2 = R^2$$

$$\cos^2\theta_1 + \cos^2\theta_2 = 1 \quad (9)$$

Therefore, by evolving the values of $\cos^2\theta_1$ and $\cos^2\theta_2$ within the range of 0 to 1 and keeping the constraint in eqn. 9, line OA is kept within the region OXY. This leads the following constraints on the parameters:

$$0 \le p_1 \le \infty$$

$$0 \le p_2 \le \infty$$

Substituting eqns. 8a and 8b into $f(p_1, p_2) = C$ to produces $f(R \cos \theta_1, R \cos \theta_2) = C$. Since the values of $\cos \theta_1$ and

$\cos\theta_2$ can be obtained from $\cos^2\theta_1$ and $\cos^2\theta_2$, the value of R can be derived to obtain the values for p_1 and p_2. If there are more than two parameters, from Pythagoras' Theorem to derive the relation between the $R\cos\theta_i$ of each parameter:

$$\sum_{i=1}^{n} p_i^2 = R^2$$

$$\sum_{i=1}^{n} R^2 \cos^2 \theta_i = R^2$$

$$\sum_{i=1}^{n} \cos^2 \theta_i = 1 \qquad (10)$$

The use of this method effectively restricts the parameters to the constraints in eqn. 11:

$$0 \le p_i \le \infty, \text{ for i = 1, 2, ..., n} \qquad (11)$$

2.3 IMPROVED FLEXIBILITY ON PARAMETER LIMITS

The method described in Section 2.2 does not allow the free setting of a limit on the parameters. Effectively, the method only allowed a lower limit of 0. In this section, the method is extended in order to provide some control over the parameter limits. The trick to achieving this is to have an alternative arbitrary point as the starting end of the R line, rather than O. This new starting position is labeled as O'.

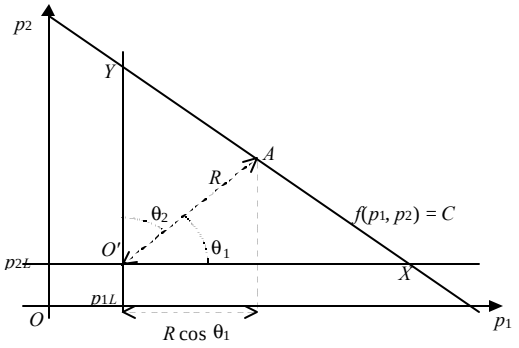


Fig. 2 Flexible Parameter Limits

To begin explaining the procedure, a two-parameter example is first considered. This example is illustrated in Fig 2. As stated earlier, the R line starts from O' instead of O. O' is located at the lower limit of the parameters involved. If $O'A$ is confined within $O'XY$, the effect is that each parameter cannot have a value lower than its lower limit. $R\cos\theta_i$ now represents the distance of a parameter from lower parameter limit p_{iL} to the intersection point. Thus, in order to obtain the final parameter value, p_{iL} is added to $R\cos\theta_i$ as shown in eqn. 12.

$$p_i = p_{iL} + R\cos\theta_i \qquad (12)$$

Considering n number of parameters and using Pythagoras' Theorem to derive the relation between the $R\cos\theta_i$ of each parameter:

$$\sum_{i=1}^{n} (p_i - p_{iL})^2 = R^2$$

$$\sum_{i=1}^{n} (R\cos\theta_i)^2 = R^2$$

$$R^2 \sum_{i=1}^{n} \cos^2 \theta_i = R^2$$

$$\sum_{i=1}^{n} \cos^2 \theta_i = 1$$

This is equivalent to eqn. 10 and thus, eqn. 10 still holds with this new definition for p_i. Substituting eqn. 12 into the constraint formula, $f(p_1, p_2, ..., p_n) = C$, the following equation is obtained:

$$f(p_{1L} + R\cos\theta_1,\ p_{2L} + R\cos\theta_2, ..., p_{nL} + R\cos\theta_n) = C \qquad (13)$$

If $\cos\theta_i$, for i = 1, 2, ..., n was provided, R can be derived in such a way that the constraint is satisfied. Since eqn. 10 still holds true with this new definition for p_i, therefore, in order for line $O'A$ to be confined to the region $O'XY$, the constraint in eqn. 10 should be satisfied during the evolution. This constraint can be summarized as:

$$\sum_{i=1}^{n} \cos^2 \theta_i = 1 \qquad (14a)$$

$$0 \le \cos^2 \theta_i \le 1, \text{ for } i = 1, 2, ..., n. \qquad (14b)$$

The constraint in eqn. 14 is a linear constraint, thereby making the design of the genetic structure simple. The resulting genetic structure would be a chromosome with $k(n-1)$ number of genes, where k is an arbitrary number chosen for resolution preference and n is the number of parameters. Each gene would have a possible integer value between 1 and n, representing the parameters that would receive the value of the packet represented by that gene. Within each chromosome, first, there would be $(n-1)$ genes representing packets of the value $\frac{2^0}{(n-1)(2^k-1)}$. These would be followed by $(n-1)$ genes for packets of value $\frac{2^1}{(n-1)(2^k-1)}$, and so on until and including $(n-1)$ genes for packets of value $\frac{2^{(k-1)}}{(n-1)(2^k-1)}$. Fig 3 below illustrates the genetic structure.

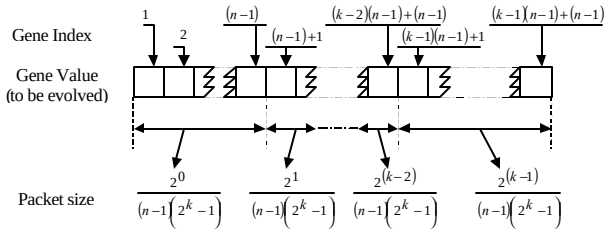


Fig. 3 Genetic Structure of Angular Transformed String

Eqn. 12 is specific for a parameter with a lower limit. An alternative to this is to consider the upper limit, where:

$$p_i = p_{iH} - R\cos\theta_i \tag{15}$$

The comparison of these two definitions is illustrated in Fig. 4. In this figure, it can be observed that:

$$p_1 = p_{1H} - R\cos\theta_1$$

$$p_2 = p_{2L} + R\cos\theta_2$$

Here the first parameter has been defined according to eqn. 15 while the second parameter is defined according to eqn. 12.

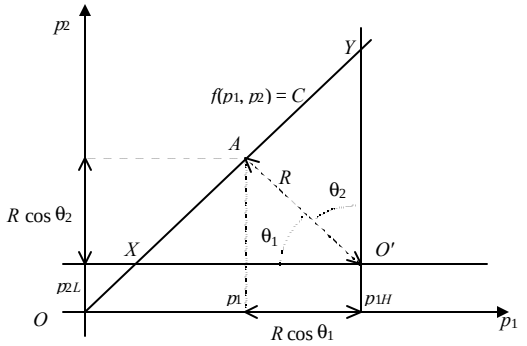


Fig. 4 Considering the Upper Parameter Limit

The purpose of having different definitions for the parameters is to avoid the event where the feasible hyper-area (formed by $f(p_1, p_2, ..., p_n) = C$) has a point at or near O'. If the hyper-area has a point at O', then the resultant R would be 0. If there is a point near O' (relative to the rest of the hyper-area), the hyper-area about this point will be better represented than the rest of the hyper-area. Thus the search is more intensive at this point compared to the rest of the feasible region. Another reason is to avoid the possibility that R is unknown. This can happen if the factors for R cancels out. The possibility of such an event may be predicted from the presence of a $\cos\theta_i$ that has a negative factor or is in a denominator. An example is the following constraint:

$$p_1 - p_2 = C$$

If $p_1 = p_{1L} + R\cos\theta_1$ and $p_2 = p_{2L} + R\cos\theta_2$, the constraint becomes:

$$(p_{1L} + R\cos\theta_1) - (p_{2L} + R\cos\theta_2) = C$$

$$R(\cos\theta_1 - \cos\theta_2) = C - p_{1L} + p_{2L}$$

Note that $\cos\theta_2$ has a negative factor. If the result of the evolution causes $(\cos\theta_1 - \cos\theta_2) = 0$, then there is no way to derive R. As an additional note, if $C - p_{1L} + p_{2L} = 0$, there is a point at O' and therefore, $R = 0$. Therefore, some caution should be used in deciding the form the parameters should take. With eqn. 12 and 15, the constraints that can be applied to the parameters are:

$$p_{aL} \le p_a, \text{ for } a \in A; \text{ and}$$

$$p_b \le p_{bH}, \text{ for } b \in B$$

where the parameters involved in the feasibility constraint are arbitrarily divided into two sets, A and B. Another point to note is that for sets of parameters with different single-sided equality constraints, this flexibility allows the simultaneous application of upper limits for certain parameters and lower limits for others.

3 PRACTICAL IMPLEMENTATION

In this section, two types of constraint optimization problems (COPs) were considered. Since angular transformation is a gene domain constraint handling method, it is compatible with objective domain constraint handling methods. As such, objective domain constraint handling methods were applied, with and without angular transformation, on the COPs. The following is a description of the objective domain methods that were considered:

1. Multi-objective method (Tan *et al*., 1999), where the constraints are assigned as additional 'hard' objectives.
2. Max penalty method, where unfeasible chromosomes are assigned a value greater than the cost of the largest cost. This is the cost variant of the zero fitness method, which is a variant of the death penalty method from (Coello Coello 1999).
3. Penalty function method (Michalewicz and Schoenauer, 1996), where the further the chromosome is from the feasible area, the higher its penalty.
4. None or control method, where no objective domain constraint handling method is used. This is used to evaluate the performance of the evolutionary algorithm should no objective domain constraint handling method be used.

The details of each constraint handling method are presented with each problem, as the specific details are adapted according to the problem. These constraint-handling methods were individually applied to a simple evolutionary algorithm (EA). Tournament selection was used with a tournament size of 2. The EA used simple two-point crossover and simple mutation, with probabilities of 0.7 and 0.01 respectively. No niching was used but switching criteria preserve strategy (SCPS) (Tan *et al*., 1999) was implemented.

3.1 PROBLEM 1

This problem is based on the nonlinear-constraint benchmark problem described in (Koziel and Michalewicz , 1999):

$$\text{Maximize } G3(P_i) = \left(\sqrt{n}\right)^n \prod_{i=1}^{n} p_i \qquad (16a)$$

subject to the constraint $$\sum_{i=1}^{n} {p_i}^2 = 1 \qquad (16b)$$

and $$0 \le p_i \le 1 \text{ for } i = 1, 2, ..., n. \qquad (16c)$$

If angular transformation were used, the logical choice would be to substitute eqn. 12 into eqn. 16b since $p_{iL} = 0$, thereby simplifying the mathematics. In order to have a variable R and to better show the angular transformation's capabilities, the constraint had been changed to:

$$\sum_{i=1}^{n} {p_i}^3 = 1$$

The cost function (eqn. 16a) was changed to a minimization problem. The constant $\left(\sqrt{n}\right)^n$ was changed such that the possible cost value is between 0 and 1 when the chromosome is feasible. Since the global solution is when all the parameters are equal, the constant shall now assume the inverse of the product of all these parameters at the global optimum. The constraint in eqn. 16c remains unchanged. The problem now becomes:

$$\text{Minimize } G3'(P_i) = 1 - \left(n\right)^{\frac{n}{3}} \prod_{i=1}^{n} p_i \qquad (17a)$$

with the constraints:

$$\sum_{i=1}^{n} {p_i}^3 = 1 \qquad (17b)$$

$$0 \le p_i \le 1 \text{ for } i = 1, 2, ..., n. \qquad (17c)$$

And the global solution is:

$$P = (p_1, p_2, ..., p_n) = \left(n^{-\frac{1}{3}}, n^{-\frac{1}{3}}, ..., n^{-\frac{1}{3}} \right) \qquad (18)$$

It should be mentioned that this problem has a nonlinear equality constraint, which is a challenge for methods using decimal coding to solve. The parameter x_i was replaced with $p_{iL} + R \cos \theta_i$. Since the lower limit p_{iL} of each parameter was 0, parameter p_i could be replaced with:

$$p_i = R \cos \theta_i \qquad (19)$$

Substituting eqn. 19 into the constraint in eqn. 17b:

$$\sum_{i=1}^{n} (R \cos \boldsymbol{q}_i)^3 = 1$$

$$R^3 \sum_{i=1}^{n} \cos^3 \boldsymbol{q}_i = 1$$

$$R = \left(\sum_{i=1}^{n} \cos^3 \boldsymbol{q}_i \right)^{-\frac{1}{3}}$$

Coding method in Fig. 3 was used to handle the following constraint (taken from eqn. 14), where the set of $\cos^2\theta_i$ was evolved.

$$\sum_{i=1}^{n} \cos^2 \boldsymbol{q}_i = 1$$

$$0 \le \cos^2 \boldsymbol{q}_i \le 1 \text{ , for } i = 1, 2, ..., n$$

The value chosen for n for the purpose of this simulation was 8. The chromosome had 42 genes, representing 42 packets. There were 6 for each of the following packet sizes:

$$\left(\frac{2^0}{7\sum_{i=0}^{5} 2^i}, \frac{2^1}{7\sum_{i=0}^{5} 2^i}, \frac{2^2}{7\sum_{i=0}^{5} 2^i}, \frac{2^3}{7\sum_{i=0}^{5} 2^i}, \frac{2^4}{7\sum_{i=0}^{5} 2^i}, \frac{2^5}{7\sum_{i=0}^{5} 2^i} \right)$$

The total value for all the packets is 1. Each gene had a possible value of 1 to 8, representing the parameters that may receive the value of the represented packet. The simulations that do not use angular transformation used simple decimal coding. Each parameter was represented by 6 genes representing the following packet sizes:

(0.1 0.01 0.001 0.0001 0.00001 0.000001)

Each gene had a value between 0 and 9, except for the genes representing the value of 0.000001, which had a value between 0 and 10. Thus, each parameter could be represented with values between 0 and 1 with a resolution of 0.000001. The value of each parameter p_i was:

$$p_i = \sum_{i=1}^{6} g_i 10^{-i}$$

where g_i = the value of the i-th gene of parameter p_i.

For the multi-objective method, there were three objectives. In the first objective, the cost was the sum of all the values of parameters exceeding the specified parameter limits. The cost value of the second objective was the deviation from the sum total in the constraint. The third objective was the objective of the simulation itself. Mathematically, these objectives are:

$$f_1(P_i) = \sum_{i=1}^{n} \begin{cases} p_{iL} - p_i & \text{if } p_i < p_{iL} \\ p_i - p_{iH} & \text{if } p_i > p_{iH} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(P_i) = \left| \sum_{i=1}^{n} p_i^3 - 1 \right| \qquad (20)$$

$$f_3(P_i) = 1 - (n)^{\frac{n}{3}} \prod_{i=1}^{n} p_i$$

where $n = 8$.

There were no goals but priority had been set. The priority for the first two objectives had been set to 1 and the third objective had a priority of 0 (don' t care).

For the max penalty method, each chromosome that does not meet any constraint (whether the parameter constraint in eqn. 17c or the overall constraint in eqn. 17b) was assigned a cost of 10 (greater than 1, the maximum possible value of a feasible chromosome). Mathematically, the cost of chromosome i is:

$$f(P_i) = \begin{cases} 10 & \text{if unfeasible} \\ 1 - (n)^{\frac{n}{3}} \prod_{i=1}^{n} p_i & \text{otherwise} \end{cases}$$

where $n = 8$

In the penalty function method, the cost of each chromosome is the sum of the cost according to the cost function, the value in excess of each parameter limit and 10 times the deviation from eqn. 17b. Mathematically, the cost of parameter i is:

$$f(P_i) = \left(1 - (n)^{\frac{n}{3}} \prod_{i=1}^{n} p_i \right) + g(p_i)$$

$$g(p_i) = \left(\sum_{i=1}^{n} \begin{cases} p_{iL} - p_i & \text{if } p_i < p_{iL} \\ p_i - p_{iH} & \text{if } p_i > p_{iH} \\ 0 & \text{otherwise} \end{cases} \right) + 10\left(\left| \sum_{i=1}^{n} p_i^3 - 1 \right| \right)$$

where $n = 8$

In the control method, where no objective-domain constraint handling method is used, the cost of parameter i is simply:

$$f(P_i) = 1 - (n)^{\frac{n}{3}} \prod_{i=1}^{n} p_i$$

For each method, the simulation had been run both with and without Angular Transformation with a population size of 100 for 100 generations. Table 1 presents the best feasible chromosome obtained.

Table 1 Results of the Simulation for Problem 1

	Angular Transformation	Decimal Coding
Multi-Objective	3.5487×10^{-4}	DNW
Max Penalty	3.5487×10^{-4}	DNW
Penalty Function	105×10^{-4}	0.6322
Control	6.0243×10^{-4}	DNW

DNW = Did not work – the simulation was unable to find any feasible chromosome.

In order to determine whether a chromosome is feasible, the chromosome was tested against the cost function in the multi-objective method (eqn. 20). If the first two objectives each have a value of 1×10^{-6} or less, the chromosome was considered feasible. It is obvious that methods that used decimal coding did not handle the nonlinear equality constraint well. The method of penalty function with decimal coding produced some results. Since the penalty was greater the further the chromosome was from the feasible region, there was some guidance for the evolution. Unfortunately, since the feasible region was small and difficult to locate, as soon as a chromosome was in the feasible region, the population converged on that chromosome. This effected the quality of the results for penalty function. When no additional constraint handling method is applied along with angular transformation, the simulation produced some results. This shows that angular transformation has effectively handled the nonlinear equality constraint.

3.2 PROBLEM 2

This is a specially designed problem with a nonlinear equality constraint that is difficult for methods using decimal coding to solve. At first it was decided that the parameters each have only lower limits and the upper limits would be derived from the available information. However, this led to a problem with p_2 and p_3 where both these parameters would have an upper limit of ∞. Decimal coding cannot represent a parameter with the range $1 \le p \le \infty$. Thus upper limits are imposed on the parameters. The problem is shown in eqn. 21:

$$Min\ G(P) = p_1(\sin(p_3\boldsymbol{p}) + \cos(p_5\boldsymbol{p})) \qquad (21a)$$

subject to $$p_1 p_2 p_3 + p_4 p_5 = 25 \qquad (21b)$$

$$1 \le p_i \le 10.9999\ ,\ i = 1, 2, 3, 4, 5 \qquad (21c)$$

Using angular transformation, the following definition is used to replace the parameters involved:

$$p_i = p_{iL} + R\cos\boldsymbol{q}_i\ ,\ \text{for } i = 1, 2, 3, 4, 5 \qquad (22)$$

Deriving R when eqn. 22 is substituted into the eqn. 21b will be difficult by conventional algebra. Therefore linear

interpolation is used instead. First, two initial values for R are selected:

$$R_1 = 0$$
$$R_2 = \frac{1}{2}\sqrt{\sum_{i=1}^{5} p_{iH} - \sum_{j=1}^{5} p_{jL}} \tag{23}$$

Each R is applied to eqn. 22 and the result is applied to the following modification of eqn. 21b:

$$p_j = p_{jL} + R_i \cos\theta_j$$
$$x_i = p_1{}^{p_2 p_3} + p_4 p_5 - 25 \tag{24}$$

where $\cos\theta_j$ is obtained by decoding the chromosome, and x_i is the error of the i-th value of R. Subsequent values for R is calculated using the following formula:

$$R_3 = \frac{R_2 x_1 - R_1 x_2}{x_1 - x_2} \tag{25}$$

This new value for R is used to interpolate subsequent values of R until an error (x_i) of less than 10^{-6} is produced or until the maximum of 50 iterations have passed. The final value for R is used to obtain the final set of parameters for the chromosome involved. Again, coding method in Fig. 3 was used to handle the following set of constraints (taken from eqn. 13):

$$\sum_{i=1}^{5} \cos^2\theta_i = 1$$

$$0 \le \cos^2\theta_i \le 1, \text{ for } i = 1, 2, 3, 4, 5$$

Each chromosome has 24 genes, representing 24 packets. There are 4 of each of the following packet sizes:

$$\left(\frac{2^0}{4\sum_{i=0}^{5} 2^i}, \frac{2^1}{4\sum_{i=0}^{5} 2^i}, \frac{2^2}{4\sum_{i=0}^{5} 2^i}, \frac{2^3}{4\sum_{i=0}^{5} 2^i}, \frac{2^4}{4\sum_{i=0}^{5} 2^i}, \frac{2^5}{4\sum_{i=0}^{5} 2^i} \right)$$

The total value for all the packets is 1. Each gene has a possible value of 1 to 5, representing the parameters that may receive the value of the represented packet. This arrangement would handle the equality constraint. However, there are still unfeasible chromosomes that violate a parameter's upper limit. For the simulations that do not use angular transformation, simple decimal coding was used. Each of the five parameters was represented by 5 genes for the following packet sizes:

(1 0.1 0.01 0.001 0.0001)

Each gene may have a value between 0 and 9. The value of each parameter p_i is:

$$p_i = \sum_{i=1}^{5} g_i 10^{(1-i)} + p_{iL}$$

where g_i = the value of the i-th gene of parameter p_i. Thus, each parameter may be represented with values between 1 and 10.9999 with a resolution of 0.0001. For the multi-objective method, there are three objectives. In the first objective, the cost is the sum of all the values of parameters exceeding the specified parameter limits. The cost value of the next objective is the deviation from the equality constraint (eqn. 21b). The last objective is the objective of the simulation itself. Mathematically, these objectives are:

$$f_1(P_i) = \sum_{i=1}^{5} \begin{cases} p_{iL} - p_i & \text{if } p_i < p_{iL} \\ p_i - p_{iH} & \text{if } p_i > p_{iH} \\ 0 & \text{otherwise} \end{cases}$$
$$f_2(P_i) = \left| p_1{}^{p_2 p_3} + p_4 p_5 - 25 \right| \tag{26}$$
$$f_3(P_i) = p_1(\sin(p_3\pi) + \cos(p_5\pi))$$

A goal of [10^{-6} 10^{-6} 10] has been set with priority [1 1 0] for objectives 1, 2 and 3 respectively. Note that a goal of 10 for objective 3 is an easily achievable goal. The first two objectives have been set as hard constraints so they would not be considered once the constraints are satisfied. With these settings, a chromosome that is feasible would dominate a chromosome that is unfeasible.

For the max penalty method, each chromosome that does not meet the constraint (eqn. 21b or eqn. 21c) is assigned a cost of 100 (a very large cost). Otherwise, the chromosome is evaluated according to eqn. 21a. Mathematically, the cost of chromosome i is:

$$f(P_i) = \begin{cases} 100 & \text{if string unfeasible} \\ p_1(\sin(p_3\pi) + \cos(p_5\pi)) & \text{otherwise} \end{cases}$$

In the penalty function method, the cost of each chromosome is the sum of the cost according to the cost function (eqn. 21a), the value in excess of each parameter limit and 10 times the deviation from the constraint in eqn. 21b. Mathematically, the cost of parameter i is:

$$f(P_i) = (p_1(\sin(p_3\pi) + \cos(p_5\pi))) + g_1(p_i) + g_2(p_i)$$

$$g_1(p_i) = \left(\sum_{i=1}^{5} \begin{cases} p_{iL} - p_i & \text{if } p_i < p_{iL} \\ p_i - p_{iH} & \text{if } p_i > p_{iH} \\ 0 & \text{otherwise} \end{cases} \right)$$

$$g_2(p_i) = 10\left(\left| p_1{}^{p_2 p_3} + p_4 p_5 - 25 \right| \right)$$

In the control method, where no objective-domain constraint handling method is used, the cost of parameter i is simply:

$$f(P_i) = p_1(\sin(p_3\pi) + \cos(p_5\pi))$$

For each method, the simulation has been run both with and without Angular Transformation with a population

size of 100 for 100 generations. Table 2 presents the best feasible chromosome.

Table 2 Results of the Simulation for Problem 3

	Angular Transformation	Decimal Coding
Multi-Objective	-16.3304	DNW
Max Penalty	-3.1526	DNW
Penalty Function	-16.3304	-1.8669
Control	-15.8336	DNW

DNW = Did not work – the simulation was unable to find any feasible chromosome.

In order to determine the feasibility of a chromosome, that chromosome is evaluated according to eqn. 26. If objective 1 and 2 each have a value of 10^{-6} or less, the chromosome is considered feasible. From the results, it is clear that decimal coding with multi-objective or max penalty was totally ineffective. Penalty function produced some results but the simulation was mainly concerned with finding a feasible chromosome rather than the optimum, hence the poor result. Many of the trials with penalty function ended up with no solution. The simulations that used angular transformation were successful in finding feasible solutions. However, the result for the control method is misleading – if the simulation was run for 200 generations with a population size of 200, it would be likely that the control method would find the global optimum that violates the parameter constraints. Thus it can be concluded from these simulations that angular transformation is capable of handling nonlinear *equality* constraints. Angular transformation can be used to handle a combined set of equality constraints. In addition to this, an objective domain method, such as multi-objective, should be used to handle the parameter constraints (which are inequality constraints) that are neglected by angular transformation.

4 CONCLUSIONS AND FUTURE WORKS

The paper has proposed an angular transformation method, which implements the feasibility constraints within the coding of chromosomes to ensure all candidate solutions to be confined within the feasible region. It can also be used to remove some of the unfeasible region in such cases, leaving a smaller search space and reduces the effort of finding the global optimum. In addition, it does not require additional parameters that are sensitive to the performance of the optimization nor extra computational effort to evaluate the unfeasible solutions. Besides, the proposed methods can be incorporated in many objective domain based constraint handling techniques to remove some of the unfeasible regions before applying these methods.

References

Coello Coello, C. A. (1999) "A survey of constraint handling techniques used with evolutionary algorithms," *Technical Report Lania-RI-99-04*, Laboratorio Nacional de Informatica Avanzada.

Hoffmeister, F., and Sprave, J. (1996) "Problem-independent handling of constraints by use of metric penalty functions," in Fogel L.J., Angeline, P.J., Back, T. (editors), *Evolutionary Programming V: Proc. of the Fifth Annual Conf. on Evolutionary Programming*, The MIT Press, Cambridge, Massachusetts, pp. 289-294.

Homaifar, A., Lai, S.H.-Y., and Qi, X. (1994) "Constrained optimization via genetic algorithms," *Simulation*, 62(4), pp. 242-254.

Joines, J. and Houck, C. (1994) "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's," *Proc. of the IEEE Int. Conf. on Evolutionary Computation*, pp. 579-584.

Koziel, S., and Michalewicz, Z. (1999) "Evolutionary algorithms, homomorphous mappings and constrained parameter optimization," *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 19-44.

Michalewicz, Z. and Schoenauer, M. (1996) "Evolutionary algorithm for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, no. 1, pp. 1-32.

Michalewicz, Z., and Attia, N. (1994) "Evolutionary optimization of constrained problems," in Sebald, A.V., and Fogel, L.J. (editors), *Proc. of the Third Annual Conf. on Evolutionary Programming*, pp. 98-108.

Schoenauer, M., and Michalewicz, Z. (1996) "Evolutionary computation at the edge of feasibility," in Ebeling, W., and Noigt, H.M. (editors), *Proc. of the Fourth Conf. on Parallel Problems Solving from Nature*, Springer Verlag, New York, pp. 245-254.

Schoenauer, M., and Xanthakis, S., (1993) "Constrained GA optimization," in Forrest, S. (editor), *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, California, pp. 573-580.

Tan, K. C., Lee, T. H. and Khor, E. F. (1999) "Evolutionary algorithms with goal and priority information for multi-objective optimization," *IEEE Proceedings of the International Congress on Evolutionary Computation*, Washington, D.C, USA, vol. 1, pp. 106-113.

Multi-Objective Mixture-based Iterated Density Estimation Evolutionary Algorithms

Dirk Thierens *dirk.thierens@cs.uu.nl*
Peter A.N. Bosman *peter.bosman@cs.uu.nl*
Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

Abstract

We propose an algorithm for multi-objective optimization using a mixture-based iterated density estimation evolutionary algorithm (MIDEA). The MIDEA algorithm is a probabilistic model building evolutionary algorithm that constructs at each generation a mixture of factorized probability distributions. The use of a mixture distribution gives us a powerful, yet computationally tractable, representation of complicated dependencies. In addition it results in an elegant procedure to preserve the diversity in the population, which is necessary in order to be able to cover the Pareto front. The algorithm searches for the Pareto front by computing the Pareto dominance between all solutions. We test our approach in two problem domains. First we consider discrete multi-objective optimization problems and give two instantiations of MIDEA: one building a mixture of discrete univariate factorizations, the other a mixture of tree factorizations. Secondly, we look at continuous real valued multi-objective optimization problems and again consider two instantiations of MIDEA: a mixture of continuous univariate factorizations, and a mixture of conditional Gaussian factorizations as probabilistic model.

1 Introduction

In classical evolutionary computation search is driven by two interacting processes: selection focuses the search to more promising points in the search space while mutation and crossover try to generate new and better points from these selected solutions. Efficient exploration requires that some information of what makes the parents good solutions needs to be transfered to the offspring solutions. If there were no correlation between the fitness of the parents and the offspring the search process would essentially be an unbiased random walk. Whether or not information is passed between parents and offspring depends on the representation and accompanying exploration operators. For mutation this is usually accomplished by letting it take small randomized steps in the local neighbourhood of the parent solution. Crossover recombines parts of two parent solutions which results in a more globally oriented exploration step. This broader exploration requires a careful choice of genotype representation and crossover operator. A common practice in the design of evolutionary search algorithms is to develop a number of representations and operators by using prior domain knowledge, and picking the best after a considerable number of experimental runs.

An alternative to this labour intensive task is to try to learn the structure of the search landscape automatically, an approach often called linkage learning (see for instance [8]). In a similar effort to learn the structure of the problem representation a number of researchers have taken a more probabilistic view of the evolutionary search process ([1, 2, 7, 9, 11, 13, 15, 14, 17]). The general idea here is to build a probabilistic model of the current parent population and learn the structure of the problem representation by inducing the dependence structure of the problem variables. The exploration operators mutation and crossover are now replaced by generating new samples according to this probabilistic model (for a survey see [16]). In [2] we have given a general algorithmic framework for this paradigm called iterated density estimation evolutionary algorithm (IDEA). In this paper we will propose an algorithm for multi-objective optimization within the IDEA framework called MIDEA. The probabilistic model build is a mixture distribution that not only gives us a powerful and computationally tractable rep-

resentation to model the dependencies in the population, but also provides us with an elegant method to preserve the diversity in the population, which is needed in order to be able to cover the Pareto front.

2 Multi-objective optimization

Optimization is generally considered to be a search process for optimal or near optimal solutions in some search space where it is implicitly assumed that given any arbitrary solution one can always tell which solution is preferred. However such a single preference criterion does not always exist. In multi-objective optimization problems different objective functions have to be optimized simultaneously. A key characteristic of multi-objective optimization problems is the existence of whole sets of solutions that cannot be ordered in terms of preference when only considering the objective function values. To formalize this we define a number of relevant concepts. Suppose we have a problem with k objective functions $f_i(\boldsymbol{x}), i = 1 \ldots k$ which - without loss of generality - should all be minimized.

1. *Pareto dominance:* a solution $\boldsymbol{x}$ is said to dominate a solution $\boldsymbol{y}$ (or $\boldsymbol{x} \rhd \boldsymbol{y}$) iff $\forall i \in \{1, \ldots, k\} : f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y}) \bigwedge \exists i \in \{1, \ldots, k\} : f_i(\boldsymbol{x}) < f_i(\boldsymbol{y})$.

2. *Pareto optimal:* a solution $\boldsymbol{x}$ is said to be Pareto optimal iff $\nexists \boldsymbol{y} : \boldsymbol{y} \rhd \boldsymbol{x}$.

3. *Pareto optimal set:* is the set $\mathcal{PS}$ of all Pareto optimal solutions: $\mathcal{PS} = \{\boldsymbol{x} \mid \nexists \boldsymbol{y} : \boldsymbol{y} \rhd \boldsymbol{x}\}$.

4. *Pareto front:* is the set $\mathcal{PF}$ of objective function values of all Pareto optimal solutions: $\mathcal{PF} = \{\mathrm{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})) \mid \boldsymbol{x} \in \mathcal{PS}\}$.

Note that the Pareto optimal set is defined in the parameter space, while the Pareto front is defined in the objective space. Multi-objective problems have been tackled with different solution strategies. A strategy which is particularly interesting from an evolutionary computation viewpoint is to search for the Pareto front - or for a representative set of Pareto optimal solutions - by making use of the Pareto dominance concept. The idea is to maintain a population of solutions that *cover* the entire Pareto front. The notion of searching a search space through maintaining a population of solutions is a key characteristic of evolutionary algorithms, which makes them natural candidates for multi-objective optimization algorithms following the *covering* strategy. The field of evolutionary multi-objective optimization has indeed seen an explosive growth in recent years (for a survey see [4]).

3 Multi-objective mixture-based IDEA

The IDEA is a framework for *Iterated Density Estimation Evolutionary Algorithms* that uses probabilistic models to guide the evolutionary search [2]. A key characteristic of this class of evolutionary algorithms is the way they explore the search space. Contrary to classical evolutionary algorithms who generate new offspring by applying crossover and mutation to individual parent solutions, IDEAs generate new offspring by sampling from a probability distribution $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}})$. The probability distribution $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}})$ is induced every generation from the $\lfloor \tau n \rfloor$ best performing individuals (n = population size, $0 < \tau < 1$). One way of achieving this, is by finding a factorized probability distribution. A factorized probability distribution is product of probability density functions (pdfs). Factorizations are usually composed either of multivariate joint pdfs or of multivariate conditional pdfs in which a single variable is conditioned on a multiple of others. The model of the probability distribution $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}})$ is determined in two steps. Firstly, a structure ς implying a factorization of the probability distribution needs to be determined. Secondly, a vector of parameters $\boldsymbol{\theta}$ need to be fitted. The choice of structure ς defines how the algorithm will explore the search space and whether it will be able to find good solutions efficiently. In this paper we will discuss four structures in the context of multi-objective optimization problems: a mixture of discrete univariate factorizations, a mixture of tree factorizations, a mixture of continuous univariate factorizations, and a mixture of conditional Gaussian factorizations. Assuming without loss of generality, we want to *minimize* $C(\boldsymbol{y})$. For every problem variable y_i we introduce a corresponding random variable Y_i and define $P^\theta(\boldsymbol{\mathcal{Y}})$ to be a probability distribution that is uniform over all vectors $\boldsymbol{\mathcal{Y}}$ with $C(\boldsymbol{y}) \leq \theta$. Sampling from $P^\theta(\boldsymbol{\mathcal{Y}})$ gives more samples that evaluate to a value below θ. To use this in an iterated algorithm, we select the best $\lfloor \tau n \rfloor$ samples in each iteration t and let θ_t be the worst selected sample cost. We then estimate the distribution of the selected samples and thereby find $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}})$ as an approximation to the true distribution $P^{\theta_t}(\boldsymbol{\mathcal{Y}})$. New samples can then be drawn from $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}})$ and be used to replace the worst $n - \lfloor \tau n \rfloor$ samples. This results in a *elitist* mechanism since we have a monotonically decreasing series $\theta_0 \geq \theta_1 \geq \ldots \geq \theta_{t_{\mathrm{end}}}$.

3.1 Factorization mixtures by clustering

The structure of the sample vector may be highly non-linear. This non-linearity can force us to use probabilistic models of a high complexity to retain some of

this non-linearity. However, especially using relatively simple probability density functions, the non-linear interactions cannot always be captured even with higher order models. The key issue is the use of *clusters*. The use of clusters allows us to efficiently break up non-linear interactions so that we can use simple models to get an adequate representation of the sample vector. Each cluster is processed separately in order to have a probability distribution fit over it. The resulting probability distribution is a weighted sum of the individual factorizations over each cluster:

$$\hat{P}_{\mathfrak{f}\langle\boldsymbol{\kappa}\rangle}(\boldsymbol{\mathcal{Y}}) = \sum_{i=0}^{|\mathfrak{K}|-1} \beta_i \hat{P}^i_{\mathfrak{f}_i}(\boldsymbol{\mathcal{Y}}) \tag{1}$$

An effective way to set the mixture coefficients β_i, is to proportionally assign larger coefficients to the clusters with a better average cost. Pelikan and Goldberg [14] proposed this method to introduce niching in the probabilistic model-building genetic algorithms. By taking the absolute value of the difference of the average cluster cost and the average initial sample vector cost, we allow for both maximization as well as minimization problems. Here we will make use of the randomized Euclidean leader algorithm which is one of the fastest clustering algorithms. The first sample to make a new cluster is appointed to be its leader. The leader algorithm goes over the sample vector exactly once. For each sample, it finds the first cluster that has a leader being closer to the sample than a given threshold $\mathfrak{T}_d$ (using the normalized Euclidean distance measure). If no such cluster can be found, a new cluster is created containing only this sample. To prevent the first clusters from becoming a lot larger than the later ones, we randomize the order in which clusters are inspected.

3.2 Multi-objective mixture-based IDEA

The algorithm discussed so far is still a single-objective optimization algorithm. To change it into a multi-objective Pareto covering optimization algorithm we need to make the following modifications:

1. First, we have to *search for the Pareto-front*: in the IDEA framework selection picks out the best $\lfloor \tau n \rfloor$ samples. Making this selection on the basis of Pareto dominance allows us to search for the Pareto front. For each individual in the population we determine the number of individuals by which it is dominated, calling this its *domination count*. All individuals are sorting according to increasing domination count and the top $\lfloor \tau n \rfloor$ solutions are selected.
2. Second, we have to *cover the Pareto-front*: maintaining diversity is needed to prevent the population to converge to a single Pareto optimal point instead of to a representative set of the entire front. Since the mixture-based IDEA already constructs a set of clusters we can simply use this to maintain the diversity. Note that the clustering can be done in the parameter space or in the objective space, but to maintain a good covering of the Pareto front clustering in the objective space is more suitable.

Finally, the multi-objective mixture-based iterated density estimation evolutionary algorithm - or MIDEA - can be summarized as:

MIDEA(n, τ)
1 Evaluate n randomly generated samples $\boldsymbol{\mathcal{P}}$
2 Iterate until termination
  2.1 Compute the domination counts
  2.2 Select the $\lfloor \tau n \rfloor$ best samples from $\boldsymbol{\mathcal{P}} \Rightarrow \boldsymbol{\mathcal{P}}^{\int}$
  2.3 Set θ_t to the worst selected cost
  2.4 Search $\boldsymbol{\mathcal{P}}^{\int}$ for a structure ς
  2.5 Estimate parameters $\boldsymbol{\theta} \xleftarrow{fit} \varsigma \Rightarrow \hat{P}_{\varsigma}(\boldsymbol{\mathcal{Y}})$
  2.6 Draw $n - \lfloor \tau n \rfloor$ new samples from $\hat{P}_{\varsigma}(\boldsymbol{\mathcal{Y}})$
  2.7 Evaluate the new samples
  2.8 Add the new samples to $\boldsymbol{\mathcal{P}}^{\int} \Rightarrow$ new $\boldsymbol{\mathcal{P}}$

Depending on the type and complexity of the application one has to choose the kind of factorization learned during the structure search. To illustrate this we will implement four versions of the MIDEA algorithm: two for discrete and two for continuous multi-objective optimization problems.

3.2.1 MIDEA_univariate and MIDEA_tree

A simple structure one can apply for discrete problems is the mixture of univariate factorizations leading to the MIDEA_univariate algorithm. The probability a problem variable has a certain value is assumed to be independent of other problem variables. Although this is a very strong assumption it appears in practice that many problems can be solved this way. When optimizing more complicated problems it is necessary to learn more structure of the domain representation. One approach which is still simple enough to be computationally efficient is to model the domain variable interactions with a tree factorization. The model thus becomes a mixture of trees, a probability model recently proposed in [12]. The MIDEA_tree can be viewed as a generalization of the optimal dependency tree algorithm [1] towards a mixture model and adapted for multi-objective problems. Interestingly, the use of a mixture of tree factorizations in proba-

bilistic model building EAs is currently also proposed in relation with the Estimation Maximization learning algorithm [17].

3.2.2 MIDEA_univariate and MIDEA_Gaussian

For multi-objective continuous function optimization we can again use a mixture model where each component distribution ignores conditional dependences between the variables. Inducing a mixture of univariate factorizations is very simple and extremely fast. A more intelligent search can be performed when using a model that learns conditional dependences between the variables. Here we induce a mixture of conditionally factorized Gaussian probability density functions. This structure has the advantage of being capable to learn conditional dependences between variables, while at the same time being computationally efficient enough to be applied at each generation. Learning a conditional factorization from the vector of selected samples can be done in a variety of ways [16]. Here we use an incremental algorithm that starts from the empty graph with no arcs. Each iteration, the arc to add is selected as the arc that increases some metric the most. If no addition of any arc further increases the metric, the final factorization graph has been found. The metric used is the Bayesian Information Criterion (for details see [3]).
Without detailed knowledge about the functions it is not possible to tell which structure is optimal. To illustrate the potential of each model we ran a number of experiments on problems previously studied in the literature.

4 Experimental results

4.1 Multi-objective 0/1 knapsack problem

Our first test function is a discrete multi-objective 0/1 knapsack problem taken from Zitzler and Thiele [18] who introduced it to compare a number of different multi-objective evolutionary algorithms. The problem is additionally interesting because of its real life practicality and the large string lengths it requires. Whereas the problem definition is relatively simple, optimizing it is difficult ($\mathcal{NP}$-hard). The multi-objective 0/1 knapsack problem consists of a set of n_I items and a set of n_K knapsacks. With each knapsack i, a weight $w_{i,j}$ and a profit $p_{i,j}$ are associated with each item j. Each knapsack i has an upper-bound c_i on the amount of weight it can hold, which is called the *capacity constraint*. The objective is to fill each knapsack so that the profit of the selected items is maximized, but without violating the capacity constraints. If item i is selected, it is automatically placed in *every* knapsack. This creates a multi-objective interaction between the knapsacks: the goal is to search for a vector of decision variables $\boldsymbol{x} \in \{0,1\}^{n_I}$ such that each objective function f_i in $(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{n_K-1}(\boldsymbol{x}))$ is maximised with

$$\forall_{i\in\{0,1,\ldots,n_K-1\}} \left[f_i(\boldsymbol{x}) = \sum_{j=0}^{n_I-1} p_{i,j}x_j \right]$$

$$\text{s.t. } \forall_{i\in\{0,1,\ldots,n_I-1\}} \left[\sum_{j=0}^{n_K-1} w_{i,j}x_j \leq c_i \right].$$

To deal with the feasibility problem with respect to the capacity constraints, we use a repair method. The type of repair method used has a great influence on the way the search space is traversed and thus on the performance of the optimization algorithm. To compare different algorithms with respect to their multi-objective performance, it is important to use the same repair method. To this end, we have used the same approach as is in [18]. If a solution violates a constraint, the repair algorithm iteratively removes items until all constrains are satisfied. The order in which the items are investigated, is determined by the maximum profit/weight ratio. The items with the lowest profit/weight ratio are removed first. This amounts to computing the quotients

$$q_j = \max_{i\in\{0,1,\ldots,n_K-1\}} \left\{ \frac{p_{i,j}}{w_{i,j}} \right\}$$

on beforehand and sorting the q_j.
The profits, weights and knapsack capacities are chosen as follows: $p_{i,j}$ and $w_{i,j}$ are random integers chosen from the interval [10,100], while the capacities c_i are set to half the items' weight in the corresponding knapsack:

$$c_i = 0.5 \sum_{j=0}^{N_I-1} w_{i,j}.$$

This results in half of the items to be expected in the optimal solutions. We performed tests on problems with two knapsacks ($n_K = 2$) allowing us to plot the Pareto front found by MIDEA and to make a visual comparison with results obtained by the Strength Pareto Evolutionary Algorithm (SPEA) and the Nondominated Sorting Genetic Algorithm (NSGA). In [18] a total of 8 algorithms were compared but for clarity we restrict ourselves here to SPEA and NSGA: they are the most commonly known and SPEA gave the best results of all 8 algorithms. Three different knapsack problems with an increasing number of items were

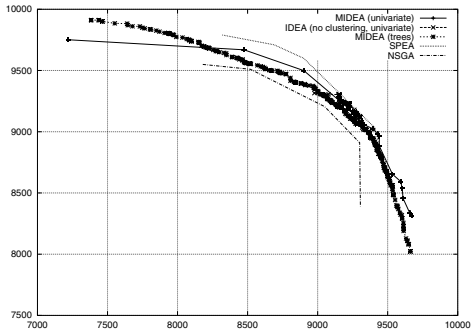


Figure 1: knapsack 250 items

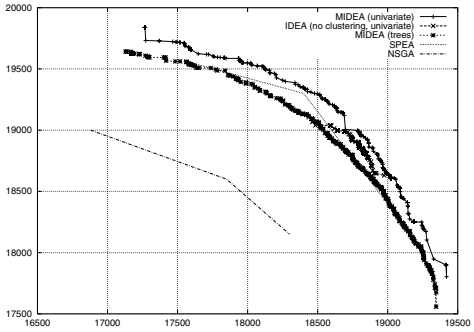


Figure 2: knapsack 500 items

studied, $n_I \in \{250, 500, 750\}$. The data sets are the same as those used by Zitzler and Thiele (available on http://www.tik.ee.ethz.ch/ zitzler/testdata.html).

In our experiments we fixed the selection size to $\lfloor \tau n \rfloor = 200$ ($\tau = 0.3$, population size $n = 667$). We have also fixed the number of evaluations to be the same as in the tests by Zitzler and Thiele, respectively $\{60000, 80000, 100000\}$ for increasing values of n_I. For the univariate factorization the final front reported is obtained by combining the results of 30 independent runs (similar to Zitzler and Thiele), but it should be noted that individual runs give an almost as wide covering of the Pareto front. Results for the tree factorization are only from one single run. The clustering is done in the objective space using the leader algorithm with $\mathfrak{T}_d$ chosen so as to get at least 5 clusters.

Figures 1, 2, and 3 show the results for the MIDEA with a mixture of univariate distributions, and for the MIDEA with a mixture of trees. To study the influence of the clustering we have also tested the univariate distributions without clustering. The graphs give also a rough sketch of the front found by SPEA and NSGA in [18]. A number of observations can be made:

1. The MIDEA algorithm found solutions with high objective function values. Performance is comparable to SPEA for the problem sets with 250 and 500 items. For the knapsack problem with 750 items MIDEA finds a Pareto front that clearly dominates the solutions found by SPEA. The results from NSGA are substantially worse.

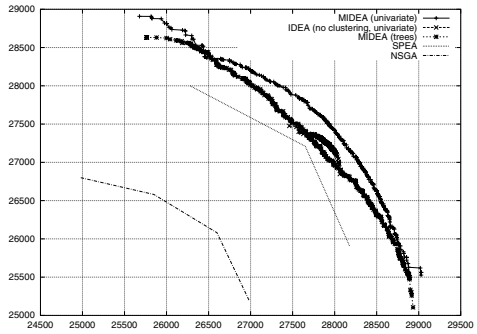


Figure 3: knapsack 750 items

2. The MIDEA algorithm found a widely covered Pareto front, wider than SPEA and NSGA.

3. Clustering is necessary for the covering to take place: without it the algorithm finds only a small part of the Pareto front.

4. On the two larger problem sets ($n_I \in \{500, 750\}$) the Pareto front obtained by the mixture of univariate distributions is slightly better than the front found by the mixture of trees. It is reasonable to assume that this is only an indication of the faster convergence speed of the mixture of univariate distributions. When more function evaluations would be done one might expect the mixture of trees algorithm to catch up, and even surpassing it for more difficult problems. This should be further investigated though.

4.2 Multi-objective continuous function optimization

Next to the multi-objective 0/1 knapsack problem we also tested the MIDEA algorithm on multi-objective continuous function optimization problems taken from the literature [5].

First we will look at the mixture of Gaussian pdfs using learning conditional factorizations. We fixed the selection size to $\lfloor \tau n \rfloor = 250$ ($\tau = 0.3$, population size $n = 834$). Clustering is done using the leader algorithm in both the objective space as well as the parameter space, with $\mathfrak{T}_d$ chosen so as to get approximately 3 clusters on the Pareto front. The final front reported is obtained by combining the results of 10

	MOP_2	MOP_4	EC_4	EC_6
Obj.	3754	10762	1019330	9535
Par.	3754	35348	2500000	8835
None	3754	43058	2500000	8426

Figure 4: Average number of evaluations.

independent runs. As before it should be noted that individual runs give an almost as wide covering of the Pareto front. The average number of required evaluations for each type of clustering is stated in figure 4. For comparison, we also tested an approach using *no* clustering. Termination is enforced when the domination count of all of the selected samples equals 0. At such a point, the selected sample vector contains only non-dominated solutions. Note that this does not have to imply at all that full convergence has been obtained since the front itself may not be optimal. To prevent the alternative of allowing an arbitrary number of generations or evaluations, a good termination criterion might be when non of the selected samples is dominated by any of the selected samples in the previous generation. For now, we restrict ourselves to the simple termination criterion, keeping in mind that premature convergence is possible. No single run was allowed more than $2\frac{1}{2} \cdot 10^6$ evaluations. Finally the conditional Gaussian factorizations are searched using the BIC metric with $\lambda = \frac{1}{2}$ (which makes this measure similar to the minimum description length measure).

Name	Objectives	Domain
MOP_2	$f_0 = 1 - e^{-\sum_{i=0}^{l-1}\left(y_i - \frac{1}{\sqrt{l}}\right)^2}$ $f_1 = 1 - e^{-\sum_{i=0}^{l-1}\left(y_i + \frac{1}{\sqrt{l}}\right)^2}$	$[-4, 4]^3$
MOP_4	$f_0 = \sum_{i=0}^{l-2} -10e^{-0.2\sqrt{y_i^2 + y_{i+1}^2}}$ $f_1 = \sum_{i=0}^{l-1} \lvert y_i \rvert^{0.8} + 5\sin(y_i^3)$	$[-5, 5]^3$
EC_4	$f_0 = y_0$ $f_1 = \gamma\left(1 - \sqrt{\frac{y_0}{\gamma}}\right)$ $\gamma = 91 + \sum_{i=1}^{l-1}\left(y_i^2 - 10\cos(4\pi y_i)\right)$	$[-1, 1] \times [-5, 5]^9$
EC_6	$f_0 = 1 - e^{-4y_0}\sin^6(6\pi y_i)$ $f_1 = \gamma\left(1 - (\frac{f_0}{\gamma})\right)$ $\gamma = 1 + 9\left(\sum_{i=1}^{l-1} \frac{y_i}{9}\right)^{0.25}$	$[0, 1]^{10}$

In figures 5 and 6, the results using objective clustering on MOP_2 and MOP_4 are shown respectively. For each of these two problems, none of the individual runs differ significantly from the combined result. Moreover, the results of parameter clustering as well as no clustering at all are also similar to these results, so we omit further graphs for these two problems. The table in figure 4 indicates that using the MIDEA algorithm requires only a few evaluations to adequately solve the two MOP problems.

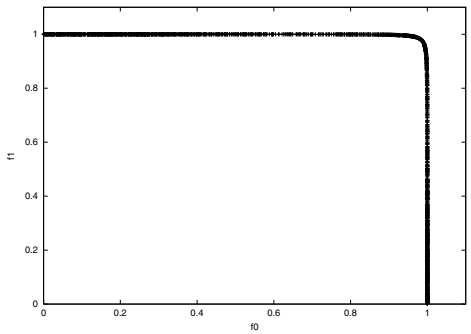


Figure 5: Result for MOP_2 (objective clustering).

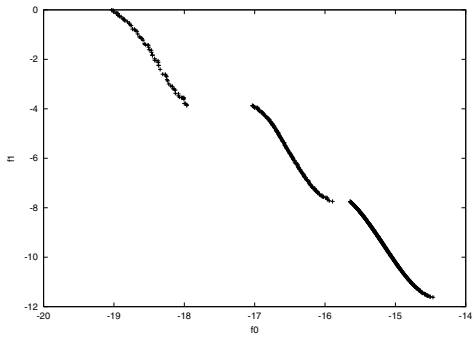


Figure 6: Result for MOP_4 (objective clustering).

Compared to EC_4, the two MOP problems are relatively simple. Converging to the optimal front is very difficult in EC_4. In figure 4, we can see that we indeed require a vastly larger number of evaluations. Only when we cluster in the objective space, do we on average require less than the maximum of $2\frac{1}{2} \cdot 10^6$ evaluations. However, closely observing the results points out that premature convergence has often taken place in the algorithm with objective clustering. Figure 7 shows the individual plots of each run. Taking more clusters in combination with a larger population size to effectively fill up and use these additional clusters, might lead to a better estimation of the promising regions of the multi-objective space. To illustrate this, we have plotted the resulting fronts after 10 runs for objective clustering with $\lfloor \tau n \rfloor = 500$ and $\mathfrak{T}_d$ such that we have approximately 5 clusters, and for parameter clustering with $\lfloor \tau n \rfloor = 125$ and $\mathfrak{T}_d$ such that we have 7 clusters. The results in figure 8 show that very good results are obtained with these settings. It should also be noted that some sort of clustering is *crucial* to be able to tackle difficult problems such as EC_4: when no clustering is applied the results are rather poor even for large populations sizes.

The main difficulty with problem EC_6 is that the optimal front is not uniformly distributed in its solutions.

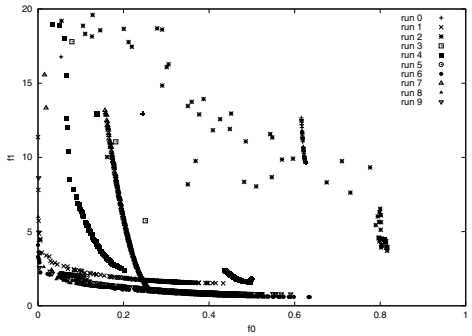


Figure 7: All runs for EC_4 (objective clustering).

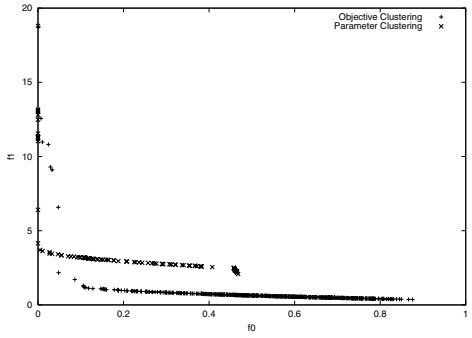


Figure 8: Results for EC_4.

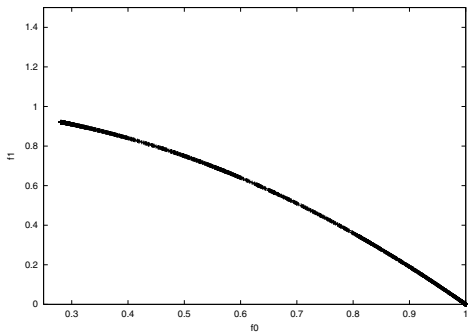


Figure 9: Result for EC_6 (objective clustering).

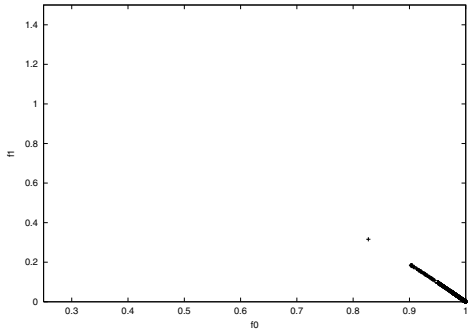


Figure 10: Result for EC_6 (parameter clustering).

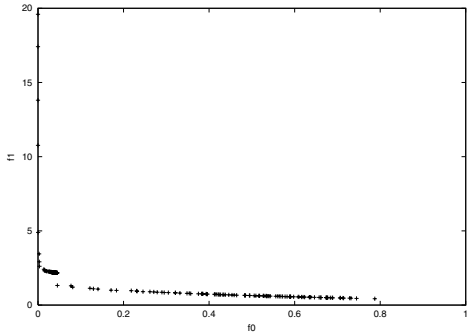


Figure 11: Result for EC_4 with univariate factorization (objective clustering).

Without clustering, we are therefore very likely to find only a part of the front. Furthermore, by clustering in parameter space, we also have no guarantee to find a good representation of the front since the parameter space is directly related to the density of the points along the front. On the other hand, clustering this space *does* give a means of capturing more regions than a single cluster can. If we are to cluster in the objective space, we should have no problem finding a larger part of the front unless the problem itself is very difficult as is the case for instance with EC_4. In figures 9 and 10, the results for objective clustering and parameter clustering are shown respectively. Using parameter clustering is clearly not effective. It should also be noted that the Pareto front found in figure 9 seems to coincide with the optimal Pareto front, which is not trivial to achieve since the fast elitist non-dominated sorting GA (NSGA-II [6]), the strength Pareto Evolutionary Algorithm (SPEA [18]), and the Pareto-archived evolution strategy (PAES [10]) are all reported to converge to a sub-optimal front ([6]).

In the experiments so far, the structure learned at each generation is a conditionally factorized Gaussian probability density function. It might well be possible that the fitness function can be optimized without the need to learn the conditional dependences between variables. In this case it would be computationally more efficient to use a probability density structure that ignores the interactions between the variables. To get a feeling of the impact of this choice we have optimized the functions EC_4 and EC_6 with a mixture of univariate factorizations. A population size of $\lfloor \tau n \rfloor = 125$, resp. $\lfloor \tau n \rfloor = 50$ ($\tau = 0.3$) was used, with a cluster threshold = 1.5, resulting in 3 to 5 clusters. Clustering was done in the objective space, and a total of 10 runs were performed. Figures 11 and 12 show that the Pareto front found is of similar quality than in the previous experiment using conditionally factorized Gaussian pdfs with objective clustering. The average

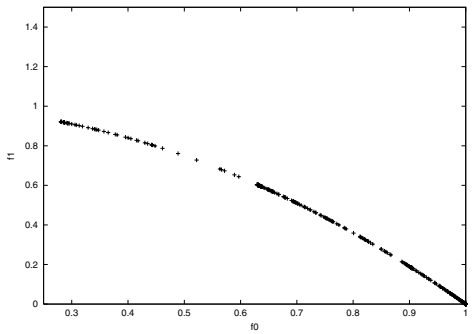


Figure 12: Result for EC_6 with univariate factorization (objective clustering).

amount of function evaluations for EC_4 was 209635, while for EC_6 the number was 2284. These figures are substantially lower than those found before (see figure 4), indicating that for these functions the computational effort spent by learning a more powerful and complicated model seems to be unnecessary. It should be noted that this gives only a rough impression about convergence speed and quality of the algorithms. Future studies will have to look at the influence of population size, selection threshold, and cluster size.

5 Conclusion

We have proposed the multi-objective mixture-based iterated density estimation evolutionary algorithm MIDEA. MIDEA builds a mixture distribution as probabilistic model resulting in a computational efficient method to represent complicated dependencies, and at the same time in an elegant procedure to search for a good covering of the Pareto front. As specific instantiations of the proposed algorithm we have implemented a mixture of univariate factorizations and a mixture of tree factorizations for discrete multi-objective optimization, and a mixture of continuous univariate factorizations and a mixture of conditional Gaussian factorizations for continuous optimization problems. Experiments showed good results for all models, including the simple univariate models.

References

[1] S. Baluja and S. Davies. Using optimal dependency–trees for combinatorial optimization: Learning the structure of the search space. In D.H. Fisher, ed., *Proc. of the 1997 Int. Conf. on Machine Learning*. Morgan Kauffman Pub., 1997.

[2] P.A.N. Bosman and D. Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, eds., *Parallel Problem Solving from Nature – PPSN VI*, pages 767–776. Springer, 2000.

[3] P.A.N. Bosman and D. Thierens. Mixed IDEAs. ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-2000/2000-45.ps.gz, 2000.

[4] C.A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[5] K. Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.

[6] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer et al., eds., *Parallel Problem Solving from Nature*, pages 849–858. Springer, 2000.

[7] M. Gallagher, M. Fream, and T. Downs. Real–valued evolutionary optimization using a flexible probability density estimator. In W. Banzhaf et al., eds., *Proc. of the 1999 Genetic and Evolutionary Computation Conference*, p. 840–846. Morgan Kaufmann, 1999.

[8] D.E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithm. In S. Forrest, ed., *Proc. of the 5th International Conference on Genetic Algorithms*, pages 56–64. Morgan Kaufmann, 1993.

[9] G. Harik, F. Lobo, and D.E. Goldberg. The compact genetic algorithm. In *Proc. of the 1998 IEEE Int. Conf. on Evolutionary Computation*, pages 523–528. IEEE Press, 1998.

[10] J. Knowles and D. Corne. The pareto archived evolution strategy: a new baseline algorithm for multiobjective optimisation. In A. Zalzala et al., eds., *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 98–105. IEEE Press, 1999.

[11] P. Larranaga, R. Etxeberria, J. Lozano, and J. Pena. Optimization by learning and simulation of bayesian and gaussian networks. TR-EHU-KZAA-IK-4-99.

[12] M. Meila and M.I. Jordan. Estimating dependency structure as a hidden variable. In M.I. Jordan et al., eds., *Proceedings of Neural Information Processing Systems*, pages 584–590. MIT Press, 1998.

[13] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evol. Comp.*, 7:353–376, 1999.

[14] M. Pelikan and D.E. Goldberg. Genetic algorithms, clustering, and the breaking of symmetry. In M. Schoenauer et al., eds., *Parallel Problem Solving from Nature*, pages 385–394. Springer, 2000.

[15] M. Pelikan, D.E. Goldberg, and E. Cantú-Paz. BOA: The bayesian optimization algorithm. In W. Banzhaf et al., eds., *Proc. of the 1999 Genetic and Evolutionary Computation Conference*, pages 525–532. Morgan Kaufmann, 1999.

[16] M. Pelikan, D.E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/99018.ps.Z, 1999.

[17] R. Santana, A. Ochoa, and M. Soto. The mixture of trees factorized distribution algorithm. *This volume*.

[18] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Interaction and Multi-objective Optimisation

Ashutosh Tiwari, Rajkumar Roy, Graham Jared and Olivier Munaux

Department of Enterprise Integration,
School of Industrial and Manufacturing Science (SIMS),
Cranfield University, Cranfield, Bedford,
MK43 OAL, United Kingdom (UK).
E-mail: a.tiwari, r.roy, g.jared, o.munaux@cranfield.ac.uk
Tel: +44 (0) 1234 754193, Fax: +44 (0) 1234 750852

Abstract

Interaction among decision variables is inherent to a number of real-life engineering design optimisation problems. The aim of this paper is to analyse multi-objective optimisation problems from the perspective of inseparable function interaction. In spite of its immense potential for real-life problems, lack of systematic research has plagued the field of interaction for a long time. The paper attempts to fill this gap by devising a formal definition and classification of interaction. It then uses this analysis as a background for identifying the challenges that interaction poses for optimisation algorithms. A number of existing test problems are also listed and analysed in this paper. The paper uses the viewpoint of inseparable function interaction developed here to devise a solution strategy and to propose an algorithm capable of handling complex multi-objective optimisation problems. The performance of the proposed algorithm is compared to that of a high performing evolutionary-based multi-objective optimisation algorithm, NSGA-II, using three test problems chosen from a set of existing problems listed and analysed in this paper. The paper concludes by giving the current limitations of the proposed algorithm and the future research directions.

1 INTRODUCTION

Real-life engineering design optimisation problems, as opposed to the theoretical problems (test cases), are those that are encountered in industry. Some examples of these problems are the design of aerospace structures for minimum weight, the surface design of automobiles for improved aesthetics and the design of civil engineering structures for minimum cost (Rao, 1996). A survey of industry and literature reveals that along with multiple objectives, constraints, qualitative issues and lack of prior knowledge, most real-life design optimisation problems also involve interaction among decision variables (Roy et. al., 2000). This interaction provides another perspective of looking at real-life design optimisation problems. Since multi-objectivity is the principal feature of most real-life problems, this paper analyses multi-objective optimisation problems from the perspective of interaction. The degree of this interaction also defines the level of difficulty of optimisation problems.

In spite of its immense potential for real-life problems, lack of systematic research has plagued the field of interaction for a long time. This can mainly be attributed to the lack of sophisticated techniques, and inadequate hardware and software technologies. However, in the last two decades, with the improvements in hardware and software technologies some research has been carried out in this area especially in the field of statistical data analysis (Draper and Smith, 1998). This has been further augmented in the recent past with the growth of computational intelligence techniques like Evolutionary Computing (EC), Neural Networks (NN) and Fuzzy Logic (FL) (Pedrycz, 1998).

2 LEVELS OF VARIABLE INTERACTION

In an ideal situation, desired results could be obtained by varying the decision variables of a given problem in a random fashion independent of each other. However, due to interaction this is not possible in a number of cases, implying that if the value of a given variable changes, the values of others should be changed in a unique way to get the required results. The two levels of interaction that can exist among decision variables are discussed below.

2.1 INSEPARABLE FUNCTION INTERACTION

The first level of interaction among decision variables, known as inseparable function interaction, is the main focus of this paper. This interaction occurs when the effect that a variable has on the objective function

depends on the values of other variables in the function (Taguchi, 1987). This concept of interaction can be understood from Figure 1. Figure 1(a) shows the case of no interaction between two variables A and B. Here, the lines representing the effect of variable A for the settings B_1 and B_2 of variable B are parallel to each other. Figures 1(b) and 1(c) show two examples of the presence of interaction. The type of interaction in Figure 1(b) is sometimes called synergistic interaction and the one in Figure 1(c) is called anti-synergistic interaction (Phadke, 1989).

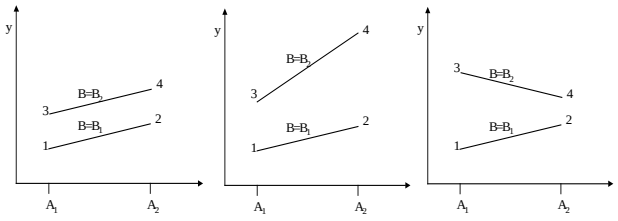


Figure 1: Examples of Interaction (a) No Interaction (b) Synergistic Interaction (c) Anti-synergistic Interaction (Phadke, 1989)

The above discussion reveals that this interaction depends on the definition of objective functions and manifests itself as cross-product terms. As an example, assume y in Figure 1 stands for A^2+B^2, having no cross-product terms. Here, y_3-y_1 is equal to y_4-y_2 (see below), making the two lines parallel and implying that there is no interaction between A and B in the given function.

$$y_3 - y_1 = (A_1^2 + B_2^2) - (A_1^2 + B_1^2) = (B_2^2 - B_1^2)$$
$$y_4 - y_2 = (A_2^2 + B_2^2) - (A_2^2 + B_1^2) = (B_2^2 - B_1^2)$$

Let us take the other case in which y in Figure 1 stands for A^2+B^2+AB, having a cross-product term AB. Here, y_3-y_1 is not equal to y_4-y_2 (see below). This makes the two lines non-parallel implying interaction between variables in the given function.

$$y_3 - y_1 = (B_2^2 - B_1^2 + A_1B_2 - A_1B_1)$$
$$y_4 - y_2 = (B_2^2 - B_1^2 + A_2B_2 - A_2B_1)$$

In GA literature, the inseparable function interaction, as defined above, is termed as epistasis. The GA community defines epistasis as the interaction between different genes in a chromosome (Beasley et. al., 1993). An alternative definition of epistasis is given by Reeves and Wright (1995), who define it in terms of alleles. In this sense, the term epistasis is used to denote the effect of a combination of alleles on the chromosome fitness that is not merely a linear function of the effects of individual alleles.

A number of real-life examples can be found in literature that involve this level of interaction. For example, the temperature (T) of an ideal gas varies with its pressure (P) and volume (V) as T=kPV, where k is the constant of proportionality. This equation has cross-product term PV clearly demonstrating the interaction between P and V in the definition of T.

2.2 VARIABLE DEPENDENCE

The second level of interaction among decision variables, known as variable dependence, occurs when the variables are functions of each other, and hence cannot be varied independently. Here, change in one variable has an impact on the value of the other. The presence of dependence among decision variables also modifies the variable search space.

Variable dependence is frequently observed in real-life problems. As an example, the resistance (R) of a wire is defined in terms of two parameters, namely Temperature (T) and Stress (S), where T and S are as defined below.

$$R = F(S, T)$$

$$T = Random(T_1, T_2)$$

$$S = f(T) + Random(S_1, S_2)$$

3 CHALLENGES FOR OPTIMISATION ALGORITHMS

Complex variable interaction poses a number of challenges for optimisation algorithms. Classical optimisation techniques, like goal programming (Charnes and Cooper, 1961), suffer from serious limitations in handling the complexity of multi-objective optimisation problems having interaction among variables. This has provided yet another motivation for the growth of research in the field of EC, NN and FL. Literature reveals that most of the multi-objective optimisation techniques are GA-based. GA' s are, therefore, the principal focus in this paper.

3.1 CHALLENGES POSED BY INSEPARABLE FUNCTION INTERACTION

GA operates on the building blocks, growing them and mixing them with each other in an attempt to solve the search problem at hand. Epistasis, termed here as inseparable function interaction, causes problems for GA by creating obstacles in the formation of these building blocks (Harik, 1997). Further, in its presence, a multi-objective optimisation problem cannot be decomposed into simpler parts. Hence, GA requires updating all decision variables in a unique way in order to maintain a spread of solutions over the Pareto optimal region or even converge to any particular solution. With a generic search operator, this becomes a difficult task for GA. Furthermore, even if a set of Pareto optimal solutions are obtained, it is difficult to maintain them since any change in one variable must be accompanied by related changes in others in order to remain on the Pareto-optimal front. The difficulties that inseparable function interaction may create for GA are summarised below (Deb, 1999).

3.1.1 Convergence to Global Pareto-optimal Front

Inseparable function interaction in objective functions may augment one or more of the following features that obstruct convergence to the true (or global) Pareto-optimal front.

- Multi-modality: In this case, GA, like many other search and optimisation methods, may converge to a local Pareto-optimal front.
- Deception: Deception is a kind of multi-modality in which almost the entire search space favours the deceptive (non-global) optimum. If present in a problem, deception misleads GA towards deceptive attractors (Goldberg et. al., 1989).
- Collateral Noise: Complex inseparable function interaction in objective functions may lead to problems that are 'rugged' with relatively large variations in the function landscape. This collateral noise may create convergence problems for GA.
- Isolated Optimum: In some problems, the optimum may be surrounded by a fairly flat search space. Since there is no useful information provided by most of the search space, GA faces difficulty in solving such problems with isolated optima.

3.1.2 Maintenance of Diverse Pareto-optimal Solutions

Maintenance of diversity in Pareto-optimal solutions may become difficult for GA due to one or more of the following features that may be enhanced in the problem by inseparable function interaction.

- Discontinuity in Pareto-optimal Front: Here the Pareto-optimal fronts are a collection of discretely spaced continuous sub-regions (Schaffer, 1984). In such problems, although solutions within each sub-region may be found, competition among them may lead to extinction of some sub-regions.
- Non-uniform Distribution over Pareto-optimal Front: In this case, feasible solutions have a non-uniform density across the Pareto-optimal front. This leads to a natural tendency for GA to find a biased distribution in the Pareto-optimal region.
- Shape Complexity of Pareto-optimal Front: Inseparable function interaction also influences the shape of Pareto–optimal front. In some cases, the shape complexity of the front may be so high that it becomes difficult for GA to find uniformly distributed solutions across it.

Further, in a number of real-life multi-objective optimisation problems, inseparable function interaction may lead to Pareto-optimal fronts that correspond to complex relationships among decision variables. All such cases become difficult for GA to handle since it is required to update the decision variables in a unique way in order to attain the desired results.

3.2 CHALLENGES POSED BY VARIABLE DEPENDENCE

As discussed in Section 2.2, the decision variables cannot be varied independently in the presence of variable dependence. Also, the search space gets modified creating a new feasible region based on the nature of dependence among decision variables. A generic GA independently varies the decision variables and works in the feasible region that does not take variable dependence into account. Hence, it creates solutions that have limited practical significance since they do not lie in the actual feasible region of the search space.

4 ANALYSIS OF EXISTING TEST PROBLEMS

A number of test problems have been reported in literature in the area of multi-objective optimisation. An analysis of these problems from the perspective of variable interaction reveals that all of them represent varying degrees of inseparable function interaction. No test problem was observed to represent variable dependence in multi-objective optimisation problems.

This section analyses the existing test problems from the point of view of interaction (Table 1). One of the test problems listed in Table 1 was cited by Veldhuizen (1999) in his work. This problem is KUR from Kursawe' s study (1990). Deb (1999) has suggested a systematic way of developing test problems for multi-objective optimisation. Zitzler, Deb and Thiele (2000) followed those guidelines and suggested six test problems. Two of these test problems, namely ZDT4 and ZDT6, have been reported in this work. Finally, the problem DEB has been directly extracted from Deb (1999) and the problem ROT from Deb et. al. (2000). These test problems were chosen with an aim to form a representative set capable of depicting the features of interaction (Section 3).

5 PROPOSED OPTIMISATION ALGORITHM

This paper focuses on the analysis of inseparable function interaction. Here an optimisation algorithm is proposed for handling complex multi-objective optimisation problems having high degrees of inseparable function interaction.

5.1 SOLUTION STRATEGY

For any continuous portion of the Pareto front, there is a unique relationship involving objective functions. This relationship is difficult to obtain analytically, and even if it is found, it has limited usefulness since mapping from function space to variable space is very complex. However, the existence of a relationship among objective functions of Pareto solutions necessarily implies that a corresponding unique relationship exists among the decision variables of these solutions.

Table 1: Analysis of Existing Test Problems (PF: Pareto Front)

Problem	n	Variable Bounds	Objective Functions (Minimisation)	Interaction Related Features
ZDT4	10	$x_1 \in [0,1]$ $x_i \in [-5,5]$ i=2,..,n	$f_1(x) = x_1$ $f_2(x) = g(x)[1-\sqrt{x_1 / g(x)}]$ $g(x) = 1+10(n-1)+\sum_{i=2}^{n}[x_i^2 - 10\cos(4\pi x_i)]$	• PF: $x_1 \in [0,1]$, $x_i=0$, i=2,..,n • Convex distribution • Multiple local fronts (21^9 or $7.94x10^{11}$) • Collateral Noise
DEB	2	[0,1]	$f_1(x) = x_1$ $f_2(x) = g(x_2)/x_1$ $g(x_2) = 2-\exp[-L_1^2]-0.8\exp[-L_2^2]$ $L_1 = (x_2-0.2)/0.004$ $L_2 = (x_2-0.6)/0.4$	• Convex distribution • Two local fronts (including global) • Deception • Isolated Optimum • Biased search space
KUR	3	[-5,5]	$f_1(x) = \sum_{i=1}^{n-1}(-10\exp(-0.2\sqrt{x_i^2+x_{i+1}^2}))$ $f_2(x) = \sum_{i=1}^{n}(\lvert x_i\rvert^{0.8}+5\sin x_i^3)$	• Non-convex distribution • Three discontinuous fronts
ZDT6	10	[0,1]	$f_1(x) = 1-\exp(-4x_1)\sin^6(4\pi x_1)$ $f_2(x) = g(x)[1-(f_1(x)/g(x))^2]$ $g(x) = 1+9[(\sum_{i=2}^{n}x_i)/(n-1)]^{0.25}$	• PF: $x_1 \in [0,1]$, $x_i=0$, i=2,..,n • Non-convex distribution • Non-uniformly spaced
ROT	5	[-0.3,0.3]	$f_1(y) = y_1$ $f_2(y) = g(y)\exp(-y_1/g(y))$ $g(y) = 1+10(n-1)+\sum_{i=2}^{n}[y_i^2 - 10\cos(4\pi y_i)]$ $y = Rx$ $R = Rotation\_Matrix$	• PF: $y_1 \in$ [Based on R], $y_i=0$, i=2,..,n • Convex distribution • Linearly related decision variables • Multiple local fronts • Collateral noise

A simple multi-objective optimisation problem is used below for explaining the above concept (Figure 2). Consider a two-objective optimisation problem having f_1 and f_2 as the two objective functions. For any continuous portion of the Pareto front, there exists a Function F involving f_1 and f_2.

$$F(f_1, f_2) = 0$$

Suppose the problem has two decision variables x_1 and x_2 that define the functions f_1 and f_2 i.e. f_1 and f_2 can be expressed as $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$. Substituting the expressions for f_1 and f_2 in the above equation yields the function F_1 in decision variables.

$$F(f_1(x_1, x_2), f_2(x_1, x_2)) = 0$$
$$\Rightarrow F_1(x_1, x_2) = 0$$

This proves the statement made earlier that a unique relationship exists among the decision variables of the solutions belonging to any continuous portion of the Pareto front. The proposed algorithm aims to explore this relationship using non-linear multi-variable regression analysis (Draper and Smith, 1998). It uses the relationship thus obtained for the following purposes.

- To perform periodic re-distribution of solutions for aiding their spread over the current front.
- To use history of change of regression coefficients for guiding the search towards global Pareto front.
- To use rate of change of regression coefficients for determining the termination condition of the algorithm.
- To re-distribute the final solutions for obtaining the whole range of well-distributed Pareto-optimal solutions.

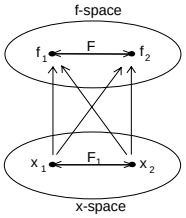


Figure 2: Solution Strategy

5.2 EPISTASIS VERSUS PROPOSED SOLUTION STRATEGY

As discussed in Section 2.1, the inseparable function interaction, defined in this paper, is termed as epistasis in

the GA community. Based on the philosophy of epistasis handling, the research in this field has taken two almost independent paths. The first path views this interaction as a race between linkage evolution and allele selection (Harik, 1997), and the second as a GA theory problem (Heckendorn and Whitley, 1999). Most of the previous research in this area has concentrated on single objective optimisation having limited number of optimal solutions. Further, this research has focused more on the theory of epistasis rather than its actual handling.

As opposed to the previous research, this paper proposes a generic solution strategy for epistasis by treating it from the point of view of definition of objective functions. This explains why epistasis has been referred to as ' inseparable function interaction' in this paper. Further, this work focuses on multi-objective optimisation problems. Since in these problems the aim is to identify as many diverse Pareto optimal solutions as possible, the formation of building blocks becomes more difficult than in single objective optimisation (Deb, 1999). The proposed solution strategy deals with this by directly targeting the Pareto front of a problem rather than its building blocks.

5.3 DESCRIPTION OF PROPOSED GENERALISED REGRESSION GA (GRGA)

The solution strategy is encoded in C++ using a new algorithm 'Generalised Regression GA (GRGA)' described in Figure 3. It should be noted that being a high performing latest algorithm, NSGA-II has been chosen as the optimisation engine for GRGA (Deb et. al., 2000). However, since GRGA is completely modular it can also be used with any other multi-objective optimisation algorithm for enhancing the algorithm performance in handling problems with complex inseparable function interaction. The steps involved in GRGA are explained below.

1. Run the optimisation cycle until all individuals have rank 0. This ensures that a front containing only non-dominated solutions is achieved. This intermediate front can be assumed to be continuous for continuous global Pareto front. As revealed in Section 5.1, regression analysis on decision variables carried out in subsequent steps of this algorithm will give relevant results only when such a continuous front is used for analysis.
2. Perform regression analysis on the decision variables to obtain the correlation coefficient (that shows how accurately the regression model represents relationship among variables) and the regression coefficients (that determine the exact nature of relationship).
3. If the correlation coefficient is greater than a pre-determined value (say 0.7), proceed to Step 4 else continue running optimisation cycle and performing regression analysis until the correlation coefficient becomes greater than 0.7. This ensures that regression analysis is used in subsequent steps only when the correlation coefficient has a value greater than 0.7. This removes the possibility of misleading the search through the use of a regression model that does not accurately represent relationship among variables.
4. If the generation number is a multiple of 10, proceed to Step 5 else go to Step 7.
5. Artificially modify the regression coefficients after every 10 generations using their history of change observed in previous generations. This guides the search towards global Pareto front by preventing it from getting trapped in local fronts.
6. Re-distribute the solutions after every 10 generations using modified regression coefficients. The aim of this step is to encourage diversity among solutions. The algorithms that can be used for re-distribution of solutions are discussed in Section 5.4.

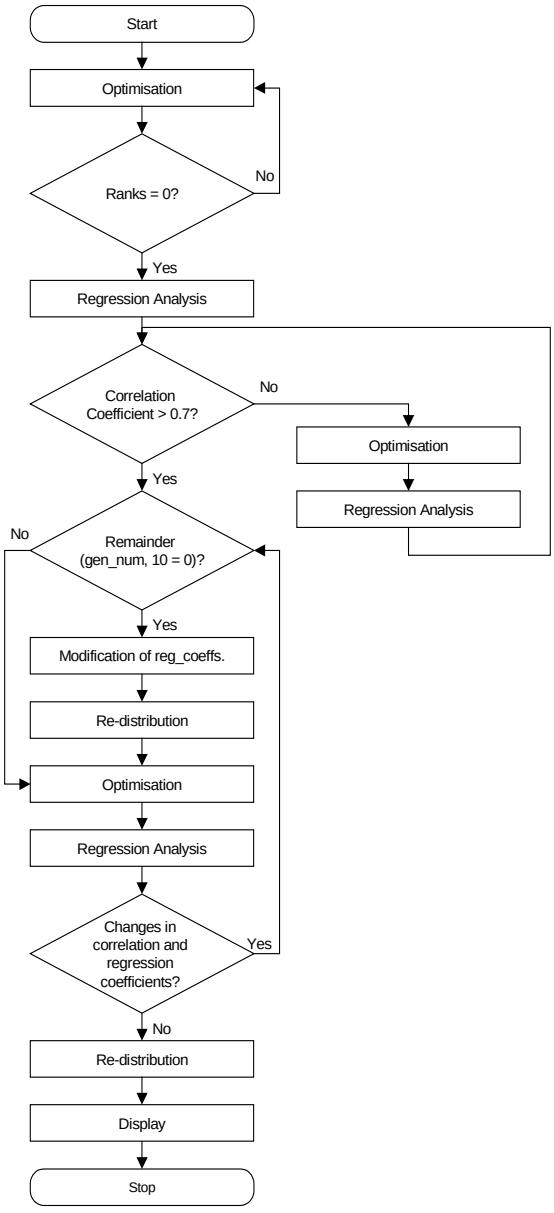


Figure 3: Generalised Regression GA (GRGA)

7. Proceed to the next generation by running the optimisation process.
8. Perform regression analysis on the decision variables.
9. If there are any changes in the values of correlation and regression coefficients in the last two generations, go to Step 4 else proceed to Step 10. No changes in the values of these coefficients imply that the Pareto front has been reached and that the algorithm should now be terminated.
10. Re-distribute the final solutions using regression coefficients. This creates solutions that are well distributed across the Pareto front.
11. Display both distributed and undistributed final solutions.

It should be noted that infinite looping is avoided in this algorithm by restricting the maximum number of generations to a pre-determined value. For the sake of simplicity, this feature is not depicted in Figure 3.

5.4 DISTRIBUTION ALGORITHMS

Distribution algorithms are used here for periodically spreading out solutions over their current front. The aim is to encourage diversity among solutions. The distribution algorithm should be able to deal with complex objective functions without significantly adding to the computational expense of the optimisation algorithm. This section proposes and analyses three different distribution algorithms.

5.4.1 Linear Distribution Algorithm (LDA)

LDA re-distributes the solutions using equally spaced decision variables in their respective ranges. This means that in a problem that has two decision variables x_1: [0:1] and x_2: [0:1], the algorithm chooses equally spaced x_1 values in [0:1] such that the number of points chosen is equal to the population size. It then uses results from regression analysis to find the x_2 values corresponding to these x_1 values. The algorithm uses this set of decision variables to form the new individuals and proceeds forward.

This algorithm is simple to implement but it works on the assumption that well-distributed points in parameter space will give rise to well-distributed points in function space. This works well for relatively simple objective functions. However, for complex functions the assumption is not valid causing the algorithm to fail.

5.4.2 Random Distribution Algorithm (RDA)

The failure of LDA in handling complex objective functions was the motivation for the development of RDA. This algorithm first generates a set of random values for x_1 in its range such that the number of points generated is equal to the population size. It then uses results from regression analysis to find the corresponding x_2 values and maps the obtained set of x_1-x_2 values back to the function space. The algorithm then determines unique points from this set and repeats the above process until the number of unique points becomes equal to the population size.

In this algorithm, a unique point is defined using the concept of diversity metric (Δ) given by Deb et. al. (2000). Here, a hypothetical circle, with radius equal to the average Euclidean distance, is drawn around the first point (Figure 4). This point is now marked as unique and all other points lying in its circle are deleted. This process is repeated till all the given points have been analysed.

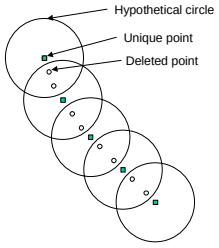


Figure 4: Identification of Unique Points

5.4.3 Hybrid Distribution Algorithm (HDA)

Although RDA has satisfactory performance, it is difficult to implement and has high computational expense. This has led to the development of HDA, in which a part of the population is generated by linear distribution and the rest is generated on a point-by-point basis using Euclidean distances between consecutive points in the function space. This algorithm is described below for a two-variable problem (Figure 5).

1. Generate equally distributed values for x_1 in its range. Number of values generated should be equal to a pre-determined proportion (say 10%) of the population size.
2. Use results from regression analysis to get corresponding x_2 values. Map the above set of x_1-x_2 values back to the function space.

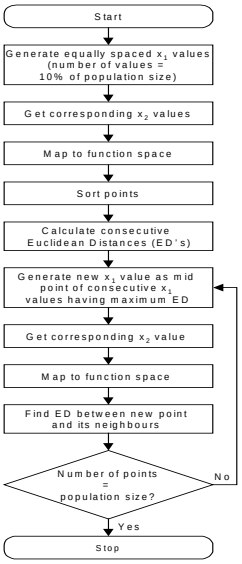


Figure 5: Hybrid Distribution Algorithm (HDA)

3. Sort these points based on function values. Find the Euclidean distances between consecutive points in the function space.
4. Generate a new x_1 value as mid-point of two consecutive x_1 values that have the maximum Euclidean distance corresponding to them.
5. Use results from regression analysis to get the x_2 value corresponding to this new x_1 value. Map the new x_1-x_2 pair back to the function space.
6. Find the Euclidean distances in the function space between this new point and its immediate neighbours.
7. Check if the total number of generated points is equal to the population size. If yes stop the process else go to Step 4.

6 PERFORMANCE ANALYSIS

GRGA was tested using three problems namely ROT, ZDT4 and ZDT6 listed in Table 1. These problems together represent a number of features that create difficulties for optimisation algorithms. Further, a number of existing multi-objective optimisation algorithms have exhibited limitations in solving these problems. This section compares the performance of GRGA with that of NSGA-II, which demonstrates better performance than most other contemporary algorithms in solving these optimisation problems (Deb et. al., 2000).

It was observed that in solving ROT the GRGA gives better distribution of solutions as compared to NSGA-II. This is illustrated in Figure 6, which compares the performance of GRGA with that of NSGA-II for 100 population size, 500 generations, 0.8 crossover probability, 0.05 mutation probability and simulated binary crossover with 10 crossover distribution index and 50 mutation distribution index. In case of ZDT4 and ZDT6, it was observed that the GRGA exhibits better convergence as compared to NSGA-II. Figure 7 compares the performance of GRGA and NSGA-II in solving ZDT6 for 100 population size, 250 generations, 0.9 crossover probability, 0.1 mutation probability and simulated binary crossover with 20 crossover distribution index and 20 mutation distribution index. It should be noted that GRGA used HDA for the tests reported here.

The tests performed with GRGA lead to the following conclusions.

- The periodic modification of regression coefficients using history of search guides the algorithm towards global Pareto front by preventing it from getting trapped in local fronts.
- The periodic distribution of solutions using regression analysis ensures that better distribution of solutions is attained across the Pareto front.
- The use of regression analysis for the termination of the optimisation cycle ensures that the process is terminated when no further improvements are possible in terms of convergence to the Pareto-optimal front.
- The re-distribution of final solutions performed in this algorithm provides the designers with the whole range of well-distributed Pareto-optimal solutions.
- As revealed in Section 4, most of the optimisation problems have varying degrees of inseparable function interaction. Since GRGA is capable of handling this interaction and the logic behind it is generic in nature, it is expected that the algorithm would perform better than the existing ones in dealing with a wide variety of optimisation problems.

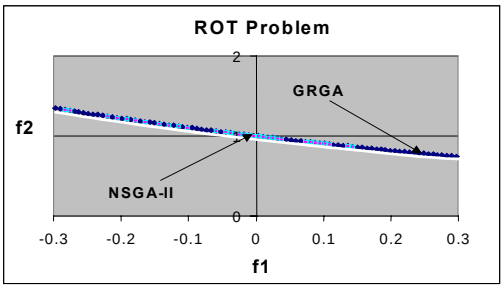


Figure 6: Performance Analysis using ROT Problem (True Pareto front shown by white line)

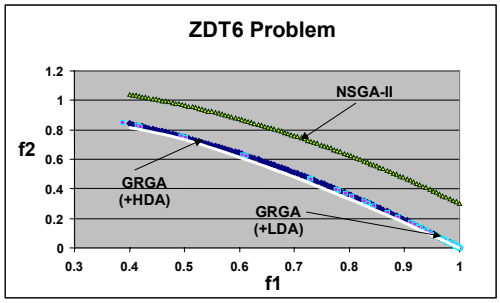


Figure 7: Performance Analysis using ZDT6 Problem (True Pareto front shown by white line)

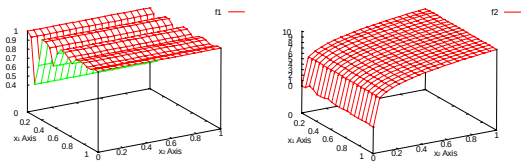


Figure 8: Objective Functions of ZDT6 (Assuming 2 variables)

A number of observations were also made regarding the performance of the three distribution algorithms discussed in Section 5.4. RDA and HDA exhibit better performance as compared to LDA especially in dealing with problems like ZDT4 and ZDT6 (Figure 7), which have complex objective functions (Figure 8). This is because LDA introduces an artificial bias in the search, leading to inferior performance in the case of complex problems. However, the computational expense of HDA is much lesser than that of RDA. Therefore, HDA has been chosen for use with GRGA.

7 FUTURE RESEARCH ACTIVITIES

The current limitations of GRGA and the corresponding future research activities are listed below.

- The algorithm in its present form requires the Pareto optimal front to be continuous. However, even if the front is discontinuous, it can, in principle, be sub-divided into continuous portions. The algorithm can then be applied individually to these portions. This provides an important area for future research.
- The performance of this algorithm is dependent on how accurately the relationship among decision variables can be represented. Hence, use of more sophisticated non-linear modelling tools like NN have the potential of improving its performance.
- The algorithm in its current form cannot deal with the second level of interaction (variable dependence). Research is currently underway to explore the use of NN for handling this interaction.
- Finally, research is going on for using sensitivity analysis to identify and maintain multiple Pareto fronts.

8 CONCLUSIONS

In spite of its immense potential for real-life problems, lack of systematic research has plagued the field of interaction for a long time. This paper proposes an algorithm capable of handling inseparable function interaction in multi-objective optimisation problems. The performance of proposed algorithm is compared to that of a state-of-the-art optimisation algorithm NSGA-II using test problems listed and analysed in this paper. It is observed that the proposed algorithm enhances the interaction handling capabilities of NSGA-II.

Acknowledgements

The authors wish to acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) – Grant No. GR/M 71473, Nissan Technical Centre – Europe (NTCE) and Structural Dynamics Research Corporation (SDRC) UK.

References

Beasley, D., Bull, D. and Martin, R. (1993). ‘ An overview of genetic algorithms: part 2, research topics’ . *University computing*, vol. 15, no. 4, 170-181.

Charnes, A. and Cooper, W.W. (1961). *Management models and industrial applications of linear programming*, vol. 1, John Wiley, New York (USA).

Deb, K. (1999). ‘ Multi-objective genetic algorithms: problem difficulties and construction of test problems’ . *Evolutionary computation*, vol. 7, no. 3, 205-230.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2000). *A fast and elitist multi-objective genetic algorithm: NSGA-II.* KanGAL Report No. 200002, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology (IIT), Kanpur (India).

Draper, N.R. and Smith, H. (1998). *Applied regression analysis.* John Wiley and Sons, Inc., New York (USA).

Goldberg, D.E., Korb, B. and Deb, K. (1989). ‘ Messy genetic algorithms: motivation, analysis, and final results’ . *Complex systems*, vol. 3, no. 5, 493-530.

Harik, G.R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* PhD. thesis, Computer science and engineering, University of Michigan (USA).

Heckendorn, R.B. and Whitley, D. (1999). ‘ Predicting epistasis from mathematical models’ . *Evolutionary computation*, vol. 7, no. 1, 69-101.

Kursawe, F. (1990). A variant of evolution strategies for vector optimization. In: Schwefel, H.P. and Manner, R. (eds.). *Parallel problem solving from nature.* Springer, Berlin (Germany).

Pedrycz, W. (1998). *Computational intelligence – an introduction.* CRC Press, New York (USA).

Phadke, M.S. (1989). *Quality engineering using robust design.* Prentice-Hall International Inc., London (UK).

Rao, S.S. (1996). *Engineering optimization – theory and practice.* Wiley-Interscience, USA.

Reeves, C.R. and Wright, C.C. (1995). An experimental design perspective on genetic algorithms. In: Whitley, D. and Vose, M. (eds.). *Foundations of Genetic Algorithms (FOGA) III*, Morgan Kaufmann, San Mateo, CA (USA).

Roy, R., Tiwari, A., Munaux, O. and Jared. G. (2000). Real-life engineering design optimisation: features and techniques. In: Martikainen, J. and Tanskanen, J. (eds.). *CDROM Proceedings of the 5th online world conference on soft computing in industrial applications (WSC5)* – ISBN 951-22-5205-8, IEEE, Finland.

Schaffer, J.D. (1984). *Some experiments in machine learning using vector evaluated genetic algorithms.* Doctoral dissertation, Vanderbilt University, Nashville, Tennessee (USA).

Taguchi, G. (1987). *System of experimental design.* Clausing, D. (ed.), UNIPUB/Kraus International Publications, vol. 1 and 2, New York (USA).

Veldhuizen, D.V. (1999). *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations.* PhD Dissertation and Technical Report No. AFIT/DS/ENG/99-01, Air Force Institute of Technology, Dayton, Ohio (USA).

Zitzler, E., Deb, K. and Thiele, L. (2000). ‘ Comparison of multiobjective evolutionary algorithms: empirical results’ . *Evolutionary computation,* vol. 8, 173-195.

A Parallel Genetic Algorithm with Adaptive Adjustment of Genetic Parameters

Naoyoshi Toshine† **Nobuyuki Iwauchi**† **Shin'ichi Wakabayashi**†
Tetsushi Koide‡ **Isao Nishimura**†
† Graduate School of Engineering ‡ Research Center for Nanodevices and Systems
Hiroshima University
4-1 Kagamiyama 1 chome, Higashi-Hiroshima 739-8527, Japan E-Mail: wakaba@computer.org

Abstract

This paper proposes a new parallel genetic algorithm with adaptive adjustment of genetic parameters, which runs on a hierarchical island model. During the execution of the parallel GA, each subpopulation executes an adaptive GA, and genetic parameters of a subpopulation with low performance are adaptively adjusted by exchanging the values of genetic parameters among the neighboring subpopulations. Experimental results show the effectiveness of the proposed algorithm compared to a parallel genetic algorithm without adaptive parameter adjustment among subpopulations.

1 Introduction

Genetic Algorithms (GAs) [4] are known as one of robust heuristic algorithms for complex large optimization problems. However, there are two notorious problems on GAs to realize their performance. One is the difficulty of setting genetic parameters to appropriate values so as to draw out a maximum capability of GA. The other is its large amount of computation time. To solve the former problem, the concept of *adaptive GAs* has been proposed [3]. An adaptive GA is a GA, in which genetic parameters are adaptively tuned automatically during the algorithm execution, and many results on adaptive GAs have been presented [6]. On the other hand, to solve the latter problem, introduction of parallel and/or distributed processing into GAs has been also intensively investigated [8].

To achieve a good performance of a GA, it is a natural idea that both parameter adaptation and parallel processing are combined in a GA to realize a high-performance parallel adaptive GA, and according to this idea, several works have already been presented. Schlierkamp-Voosen and Mühlenbein proposed the breeder genetic algorithm (BGA) [13], in which adaptation is performed among subpopulations. Although the authors did not call their GA the parallel GA, it could be run in a parallel/distributed environment. Schnecke and Vornberger proposed a parallel GA for the combined optimization of placement and routing in VLSI layout [11]. The main idea of this algorithm is the self-adaptation of the search process. Several islands execute a sequential GA with different strategies. At fixed intervals, these strategies are ranked and each strategy is adjusted to the next better one by assimilating its characteristical parameters. In contrast to those subpopulation based adaptive parallel GAs, Budin *et al.* proposed a parallel adaptive GA, in which there exists only a single mating pool, but a number of threads can operate on the population at the same time, each one acting independently [1]. The adaptive method determines the way in which the genetic operators applied, not interfering with the operators themselves. Recently, Sawai and Adachi presented a parallel GA, which relieved the user from having to set the parameters of the GA [10]. In their GA, the search strategy was based on a dynamic change of subpopulation size extracted from the population.

From the implementation point of view, a parallel adaptive GA based on an island model, in which the population consists of some number of subpopulations, is preferred, especially when the GA is implemented on a network of workstations. As mentioned above, several parallel adaptive GAs based on an island model have been proposed. In this paper, we have several concerns on parallel island model GAs with parameter adaptation. First, the way how to realize and control parameter adaptation is a crucial point to design a parallel adaptive GA. The adaptation mechanism may be defined by the set of parameters to be adapted, adaptation timing, the method of updating parameters, and so on. Second, the ranking method to evaluate each subpopulation is important. When adapting the parameters in each subpopulation, usually, we should evaluate the whole set of parameter values of that subpopulation, and based on that, parameter values are exchanged among subpopu-

lations to update them. Third, we are interested in how to choose initial values of parameters, which are both ordinary GA parameters and "strategy" parameters to control the parameter adaptations, in each subpopulation. Finally, in [13], Schlierkamp-Voosen and Mühlenbein pointed out that the crucial question of the adaptation based on subpopulations concerned the level where the adaptation was done, and the level of subpopulations or the level of populations could be used. However, until now, this problem has not been discussed further in detail.

In this paper, we propose a new parallel GA with adaptive adjustment of genetic parameters based on an extended island model. The main features of the proposed GA are as follows. First, we introduce several adaptation control parameters so that the behavior of each subpopulation can be characterized differently each other. Based on those parameters, second, we also propose a new adaptation algorithm among subpopulations. Third, we propose a new strategy to initialize the values of genetic parameters in each subpopulation. Fourth, we investigate several methods for ranking subpopulations. Finally, experimental results are provided to show the effectiveness of the proposed adaptive GA.

The paper is organized as follows: It starts with a description of the overall algorithm. Then, the proposed adaptation method is presented. Finally, experimental evaluation on the proposed algorithm is described.

2 Preliminaries

2.1 Population model

The proposed parallel adaptive GA adopts an extended island model as its population model. An island parallel GA is a GA, in which a whole population is divided into a set of mutually disjoint subpopulations, and for each subpopulation, a process is assigned to execute a GA procedure. During the execution of the GA, each process exchanges the individual data to prevent a premature convergence to a local optimum. This operation is generally called *migration*. The migration process can be characterized by its timing and the selection and update algorithm of migrated individuals. For migration timing, it can be done synchronously or asynchronously. Synchronous migration is the one, in which all subpopulations synchronously exchange their individual data. In case of asynchronous migration, each subpopulation independently starts migration when a given condition is satisfied in the subpopulation. For the selection and update algorithm of migrated individuals, there are also several variants. When a subpopulation receives a request to send an individual data, there are several selection methods to choose it, for example, the best one, or an arbitrary one. When a subpopulation receives a migrated data from another subpopulation, there are also several update methods, for example, to add it into the subpopulation no matter how its fitness value is, or to add it only when the migrated data is superior than any data in the current subpopulation.

An island parallel GA is also characterized by the communication topology among subpopulations. This is often represented by a graph, in which each vertex represents a subpopulation, and there is an edge between subpopulations if migration is performed between those two subpopulation. If a subpopulation A has an edge to a subpopulation B, we say A is a *neighbor* of B.

In this paper, we adopt an extended island population model as the population model of the proposed parallel adaptive GA. The basic structure of our model is the same as the ordinary island model. However, we introduce a hierarchical structure into the island model. In our model, a set of subpopulations is divided into a set of mutually disjoint groups of subpopulations, and we call each group a *community*. Mathematically speaking, a set of subpopulations can be regarded as a refinement of a set of communities. The aim of introducing the concept of communities is to control the parameter adaptation among subpopulations. We have a hypothesis that, to achieve a good performance of a parallel adaptive GA, it is more effective to set parameter values in neighboring subpopulations to similar values rather than to set them to totally different values. According to this hypothesis, as mentioned below, each community will have the same initial values for some genetic parameters. The validity of this hypothesis is shown by experimental results.

Figure 1 shows an example of our proposed population model, in which there are 16 subpopulations, and each 4 subpopulations constitute a community. In this example, subpopulations in a community are more tightly connected with each other than with subpopulations outside the community so that parameter adaptation will mainly be performed in each community.

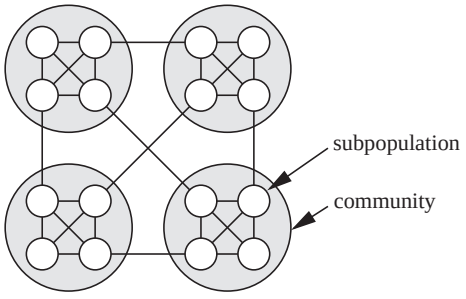


Figure 1: An example of the proposed population model.

2.2 Adaptive GA for each subpopulation

Parameter adaptation of a parallel adaptive GA may be classified into two types: parameter adaptation among subpopulations, and parameter adaptation in each subpopulation. Although the former is more interesting from the viewpoint of parallel GAs, the latter is still worth considering to get a better GA performance. There have been many adaptive GAs so far, and many of those could be used as the adaptive GA procedure for each subpopulation in the proposed parallel adaptive GA. In this paper, we adopt an elite-degree based adaptive GA (EAGA), which we have proposed in [5]. An EAGA is a generational adaptive GA, in which, when recombining a pair of individuals, a potential superiority of each individual is estimated by a special measure called *elite degree*, and based on their elite degrees, types of crossover operators and the values of mutation probabilities are selected and applied to individuals. An individual having a high elite degree is supposed to have many good schemata, and so a schema preserving crossover such as 2-point crossover is applied. Otherwise, for an individual having a low elite degree, a more disruptive crossover such as uniform crossover is applied to explore the search space widely. Selecting GA operators and parameter values for each individual based on elite degree, an EAGA can realize a good balance of global search and local search so that a good solution can be found in a short computation time. For formal description of elite degrees, see Appendix.

3 Parallel Adaptive GA

3.1 Outline of the algorithm

The outline of the proposed algorithm is as follows. First, initial genetic parameter values are randomly set in each subpopulation. During the execution of the algorithm, each subpopulation executes the adaptive GA procedure [5] to evolve the individuals. In this process, the crossover operators and mutation probabilities are adaptively selected to each individual based on elite degree. Migration to exchange individual data among subpopulations is also executed when a given migration condition is satisfied. After the execution of GA procedures for one generation, each subpopulation checks whether the condition to start parameter adjustment among subpopulations is satisfied. Adaptive parameter adjustment to genetic parameters is performed among neighboring subpopulations if no best individual has been updated in the last k generations by genetic procedures in the subpopulation except migration, where k is a user-specified parameter. Then, genetic parameters of the subpopulation are updated according to the ones in a neighboring subpopulation, whose performance is ranked to be the best among neighboring subpopulations. After the parameter adjustment among subpopulations, the adaptive GA procedure in a subpopulation is started again. This process is repeated until the specified number of generations are executed. The flowchart of the proposed algorithm concerning each subpopulation is shown in Figure 2.

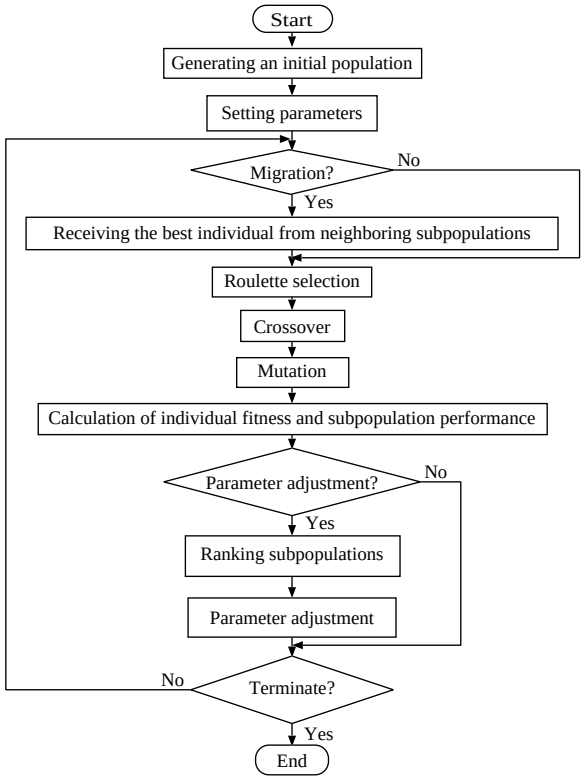


Figure 2: Outline of the proposed algorithm.

3.2 Genetic parameters to be adjusted

In the proposed algorithm, genetic parameters are classified into four types, that is, adjustment parameters, subpopulation parameters, community parameters, and population parameters. Adjustment parameters are the parameters to be adaptively adjusted during the algorithm execution, and those initial values are set independently and randomly in each subpopulation. Another parameters are static parameters, and those values are specified at the start of the algorithm execution and fixed during the algorithm execution. Subpopulation parameters are parameters, for which initial values are set independently in each subpopulation. Community parameters are the parameters, whose values are common among subpopulations in a community. Population parameters are parameters to be used to control the overall behavior of the algorithm, and those values are common in all subpopulations. In the following, parameter values and their ranges of values are listed.

1. Adjustment parameters
   - crossover probability† $P_c \in [0.0, 1.0]$
   - the number of crossover points for elite crossover† $CP \in \{1, 2, 3, 4, 5\}$
   - low mutation probability† $P_{mL} \in [0.0, 0.05]$
   - high mutation probability† $P_{mH} \in [0.0, 0.5]$
   - threshold for adaptive operator selection $D_{th} \in [0.0, 4.0]$
   - migration rate‡ $R_m \in \{1, 2, \ldots, 100\}$
   - migration exchange mode‡ $M_{exc} \in$ {asynchronous, synchronous, random}
   - migration update mode‡ $M_{up} \in$ {conditional, non-conditional}
2. Subpopulation parameters
   - adjustment weight* $W_{adj} \in [0.0, 1.0]$
   - adjustment start generation* $T_{adj} \in \{0, 500, 1000, 1500\}$
   - adjustment restart interval* $I_{adj} \in \{100, 101, \ldots, 500\}$
   - adjustment rejection rank* $R_{rej} \in \{\emptyset, \{2\}, \{2, 3\}\}$
   - adjustment order* O_{adj}
3. Community parameters
   - subpopulation size $N \in \{30, 31, 32, \ldots, 100\}$
   - elite decision factor† $\alpha \in [0.0, 1.0]$
   - elite influence factor† $\beta \in [0.0, 1.0]$
4. Population parameters
   - the maximum number of generations T_{max}
   - the number of subpopulations in a community N_{sub}
   - the number of communities N_{com}
   - initial setting mixture ratio R_{ini}

In the above list, parameters with † are parameters used in the adaptive procedure based on elite degree proposed in [5]. We assume that, in this paper, for elite crossover operator, one of 1-point, 2-point, 3-point, 4-point, and 5-point is adaptively selected to be applied to each pair of elite individuals. For recombining non-elite individuals, uniform crossover is used. Parameters with ‡ are parameters to be used to control the individual data exchange among subpopulations. For lack of space, the details of them are omitted here. For detailed description on those, please refer to [12]. Parameters with $*$ are parameters to control adaptive parameter adjustment among subpopulations, that will be explained in the following.

3.3 Initial parameter setting

Performance of a parallel GA partly depends on the initial values of genetic parameters. In [7], Miki *et al.* showed that setting different values for each subpopulation was more effective than setting the same values in an island parallel GA. On the other hand, if we randomly generate an initial value to each parameter, a parameter may have an inappropriate value. Thus, it is better to restrict the range of the value of each parameter.

In this paper, initial genetic parameter setting is performed in two different ways, called *standard setting* and *non-standard setting*. In the standard setting, an initial value of a parameter is randomly generated within a narrow range, whose center value is equal to the standard one. The standard value of each parameter is derived from the De Jong's standard parameter values or the ones which are empirically shown to be best effective. On the other hand, in the non-standard setting, an initial value is randomly generated within a more wider range.

In the proposed parallel adaptive GA, initial values of adjustment parameters are set by both the standard and non-standard setting methods, and for each subpopulation, one of two initial setting methods is randomly selected according to the initial setting mixture ratio R_{ini}. For example, if $R_{ini} = 0.8$, then for 80% of subpopulations, the standard setting is applied. Table 1 shows the ranges of parameter values. For subpopulation and community parameters, only the standard setting is applied to produce the initial values of parameters. For population parameters, the user specifies them.

Table 1: Standard and non-standard parameter setting

	standard setting	non-standard setting
P_c	$0.4 \sim 0.8$	$0.0 \sim 1.0$
CP	$2, 3$	$1, 2, 3, 4, 5$
P_{mL}	$0.0001 \sim 0.005$	$0.00001 \sim 0.05$
P_{mH}	$P_{mL} \times 10$	$0.0001 \sim 0.5$
R_m	$5 \sim 40$	$1 \sim 100$
M_{exc}	async., sync., random	async., sync., random
M_{up}	cond., non-cond.	cond., non-cond.
D_{th}	$2.0 \sim 3.5$	$0.0 \sim 4.0$

3.4 Parameter adjustment order

In the proposed algorithm, not all adjustment parameters are updated at a time. Adjustment parameters are classified into some number of groups, and at a time when the adaptive parameter adjustment is performed, parameters in one group are updated. In the current implementation of the algorithm, adaptive adjustment parameters are classified into the following four groups: $G_1 = \{P_c, CP\}, G_2 = \{P_{mH}, P_{mL}\}, G_3 = \{R_m, M_{exc}, M_{up}\}$,

and $G_4 = \{D_{th}\}$. The order of groups to be adjusted is given by the adjustment order O_{adj}, which is a subpopulation parameter.

3.5 Ranking subpopulations

When a subpopulation starts the parameter adjustment with its neighboring subpopulations, first, it determines the rank of each subpopulation based on some criterion to evaluate the performance of the current parameter set. We have performed some preliminary experiments for 8 kinds of criterion [12], and from the experiments, we adopt the following three methods as the ranking methods. We assume that the method to be actually used in the proposed algorithm is specified in advance by the user.

- the diversity of genotypes [2]
- the standard deviation of fitness values [9]
- the number of times the best fitness has been updated.

3.6 Adaptive parameter adjustment among subpopulations

Adaptive parameter adjustment among subpopulations in the proposed parallel adaptive GA consists of three phases described below. Let T be the current generation, and I be the number of generations passed since the latest generation, in which the best individual has been updated without migration. Parameter adjustment will be started if both $T \geq T_{adj}$ and $I \geq I_{adj}$ are satisfied, where T_{adj} and I_{adj} are both subpopulation parameters. It means that, parameter adjustment in a subpopulation will happen when a subpopulation seems to have a low possibility to find a better individual with the current parameter values.

3.6.1 Phase 1

In Phase 1, the ranks of neighboring subpopulations are determined. Let p be a subpopulation, for which the adaptive parameter adjustment is performed, and let $NB(p)$ be the set of neighboring subpopulations of p including p itself. For each $q \in NB(p)$, let $R(q)$ be the rank of q, and $R_{last}(q)$ be the rank of q in the last parameter adjustment. The rank is calculated by the user-specified criterion described in Subsection 3.5.

3.6.2 Phase 2

In Phase 2, the algorithm determines whether to start parameter adjustment or not. If the rank of p itself is 1, then it may be difficult to change its situation by receiving the parameter values from other subpopulations. Thus, p changes its parameter values by itself. Otherwise, parameter adjustment based on the parameter values of other subpopulation will be started. Detailed description of Phase 2 is as follows.

Step 1 If $R(p) > 1$ then go to Phase 3. Otherwise, if $R_{last} = 1$, then go to Step 2, else terminate the parameter adjustment.

Step 2 If all adjustment parameters have been adjusted, then go to Step 3, otherwise go to Step 4.

Step 3 Set all adjustment parameters to the initial values, which were set at the start of the algorithm. Terminate.

Step 4 Set the parameter values using the setting method, which is different from the one used in the last parameter setting. For example, if, in the last setting, the standard setting was applied, then the non-standard setting will be applied.

3.6.3 Phase 3

In Phase 3, parameter adjustment among subpopulations is actually performed. Characteristics of each subpopulation for parameter adjustment may be different. For example, using parameter R_{rej}, some subpopulation may be set to be persistent to keep its parameter values. The order of parameters to be adjusted may be also different among subpopulations.

Step 1 If $R(p) \in R_{rej}$ then terminate. Otherwise, go to Step 2.

Step 2 Find the subpopulation $q \in NB(p)$ whose rank is 1.

Step 3 Obtain the values of adjustment parameters in a group specified by the parameter adjustment order O_{adj} from q.

Step 4 Compare the parameter values obtained in Step 3 with the current parameter values of p. If the difference between the obtained value and the current value is small, then go to Step 5. Otherwise, go to Step 6.

Step 5 If all groups of adjustment parameters have been updated according to the adjustment order, then let q be the subpopulation with the next rank of current q, and go to Step 3. Otherwise, go to Step 3.

Step 6 Let $I(q)$ be the number of generations passed since the latest generation, in which the best individual has been updated without migration in q. If $I(q) \leq I_{adj}$, then go to Step 7, else go to Step 8.

Step 7 For each adjustment parameter, denoted r, in the current group to be adjusted in p, let r' be the corresponding parameter in q. Then, $r \leftarrow r + W_{adj} \times (r - r')$, where W_{adj} is the adjustment weight.

Step 8 $r \leftarrow r' \times (0.8 + rand(0.4))$, where $rand(t)$ is a random number in $[0.0, t]$.

4 Experiments

The proposed parallel adaptive GA has been implemented with the C language and the PVM (Parallel Virtual Machine), a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource, running on two workstations (Ultra COMPstation Model 170×2) connected with a LAN. We used the following three benchmark functions to evaluate the proposed GA.

$$f_{11}(\vec{x}) = 20 \times 10 + \sum_{i=1}^{20}(x_i^2 - 10\cos(2\pi x_i))$$
$$-5.12 \leq x_i \leq 5.12 (i = 1, 2, \ldots, 20)$$
$$\min(f_{11}) = f_{11}(0, 0, \ldots, 0) = 0$$
$$f_{13}(\vec{x}) = 1 + \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10}(\cos(\frac{x_i}{\sqrt{i}})$$
$$-600 \leq x_i \leq 600 (i = 1, 2, \ldots, 10)$$
$$\min(f_{13}) = f_{13}(0, 0, \ldots, 0) = 0$$
$$f_{14}(\vec{x}) = \sum_{i=1}^{50} f'(x_i)$$
$$(x_i, f'(x_i)) = ((0, 3), (1, 2), (2, 1), (3, 0), (4, 4))$$
$$(i = 1, \ldots, 50)$$
$$\max(f_{14}) = f_{14}(4, \ldots, 4) = 200$$

In the experiments, the population parameters were set as follows: $T_{max} = 3000$, $N_{com} = 4$, $N_{sub} = 4$ and $R_{ini} = 0.5$. The population model used in the experiments is the one shown in Figure 1. The proposed algorithm was compared with an algorithm without parameter adaptation among subpopulations, that is, the algorithm was the same behavior as the proposed one except that no parameter values were exchanged among subpopulations. Note that, in each subpopulation, adaptive parameter selection based on elite degree [5] was performed in the both algorithms. Two algorithms were run in 20 times with the different initial values of individual data and genetic parameters. In each run, the same initial values of corresponding parameters of two algorithms were set. As the methods of evaluating a subpopulation, the three methods described in Subsection 3.5 were used.

Results of the experiment were summarized in Tables 2, 3, and 4. In Table 2, the average, the best, and the worst of the best fitness in 20 runs were shown. From the table, as the best value, the proposed algorithm produced the better results than the non-adaptive method in most cases. As the evaluation method of the performance of a subpopulation, the method based on the diversity of genotypes [2] may be the best among the three. Next, in Table 3, we evaluated the effectiveness of the proposed hierarchical population model by changing the number of communities. In this experiment, we used f_{11} as the benchmark function, and the diversity of genotypes was used to evaluate the subpopulation performance. From Table 3, we see that the hierarchical model was in fact effective. In this experiment, the topology of 4×4 was the best. Finally, we changed the initial setting ratio, and compared the results. Table 4 showed the results. From the table, it is appropriate to set the ratio of nonstandard setting to 0.5 to 0.75. We have performed several other experiments using the different benchmarks, and have obtained the similar results. Thus, we conclude that the proposed parallel adaptive GA is in fact effective to obtain good results.

Table 3: Evaluation of the proposed hierarchical population model.

$N_{com} \times N_{sub}$	average	best	worst
16×1	35.0	12.8	69.7
8×2	37.2	7.8	71.4
4×4	31.0	4.5	65.9
2×8	24.6	10.0	52.6
1×16	33.4	8.3	85.2

Table 4: Evaluation of the initial setting method.

stand. : non-stand.	average	best	worst
100% : 0%	41.5	11.8	71.6
75% : 25%	39.8	16.6	70.8
50% : 50%	33.8	4.5	65.9
25% : 75%	28.9	7.7	67.3
0% : 100%	24.3	8.9	44.9

5 Conclusion

In this paper, we proposed the new parallel adaptive GA, in which parameter values were exchanged to adjust them. Experimental results show that the proposed parallel adaptive GA can produce better solutions than previous methods. In particular, introducing a hierarchical structure into the population model has been shown to be effective. Future research includes the development of a more effective adaptive method to adapt parameter values among subpopulations.

Table 2: Comparison with the non-adaptive parallel GA.

function	the evaluation method of a subpopulation	average	best	worst
f_{11}	diversity in a genotype	31.0	4.5	65.9
	standard deviation of fitness	35.1	8.1	71.9
	the number of updates of the bests	30.3	3.7	75.2
	non-adaptive GA	28.4	11.5	54.7
f_{13}	diversity in a genotype	0.492	0.067	1.289
	standard deviation of fitness	0.629	0.045	1.950
	the number of updates of the bests	0.702	0.069	1.887
	non-adaptive GA	0.630	0.084	1.501
f_{14}	diversity in a genotype	28.0	11.0	52.0
	standard deviation of fitness	25.3	15.0	42.0
	the number of updates of the bests	29.5	17.0	49.0
	non-adaptive GA	29.6	17.0	52.0

Acknowledgments

This research was supported in part by Grant-in-Aid for Scientific Research (C)(2)(No.12838008) from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] L. Budin, M. Golub, and D. Jakobovic, "Parallel adaptive genetic algorithm," *Proc. International ICSC/IFAC Symp. on Neural Computation*, pp.157–163 (1998).

[2] R. J. Collins, and D. R. Jefferson, "Selection in massively parallel genetic algorithms," *Proc. International Conference on Genetic Algorithms*, pp.249–256 (1991).

[3] L. Davis, "Adapting operator probabilities in genetic algorithms," *Proc. International Conference on Genetic Algorithms*, pp.61–69 (1989).

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley (1989).

[5] K. Hatta, K. Matsuda, S. Wakabayashi, and T. Koide, "On-the-fly crossover adaptation of genetic algorithms," *Proc. Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp.197–202 (1997).

[6] R. Hinterding, Z. Michalewicz, and A. E. Eiben, "Adaptation in evolutionary computation: A survey," *Proc. IEEE International Conference on Evolutionary Computation*, pp.65–69 (1997).

[7] M. Miki, T. Hiroyasu, M. Kaneko, and K. Hatanaka, "A parallel genetic algorithm with distributed environment scheme," *Proc. IEEE Systems, Man and Cybernetics Conference*, pp.695–700 (1999).

[8] H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," *Proc. International Conference on Genetic Algorithms*, pp.416–421 (1989).

[9] M. Munetomo, Y. Takai, and Y. Sato, "An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithm," *Proc. International Conference on Genetic Algorithms*, p.649 (1993).

[10] H. Sawai, and S. Adachi, "Parallel distributed processing of a parameter-free GA by using hierarchical migration methods", *Proc. Genetic and Evolutionary Computation Conference*, pp.579-586 (1999).

[11] V. Schnecke, and O. Vornberger, "An adaptive parallel genetic algorithm for VLSI layout optimization," *Proc. Parallel Problem Solving from Nature IV*, pp.859–868 (1996).

[12] N. Toshine, "A parallel genetic algorithm with adaptive adjustment to genetic parameters on a new multi-population model," *Master Thesis*, Graduate School of Engineering, Hiroshima University (2000).

[13] D. Schlierkamp-Voosen, and H. Mühlenbein, "Strategy adaptation by competing subpopulations", *Proc. Parallel Problem Solving from Nature III*, pp.199-208 (1994).

Appendix

A Elite Degree

In [5], we introduced a new measure called the *elite degree*, which shows how much the good schema is supposed to be included in an individual (chromosome). Let the generation of initial population be generation 0, and $T(> 0)$ be the current generation. Let $x_i^T (1 \leq i \leq N$, where N represents the population size) be the i-th individual of the population in the generation T, and $Anc_i^T\ (j)$ be the set of ancestors in $T - j$ generation of individual x_i^T.

In order to define the elite degree, first, we define an *elite*. Consider the case of the maximization problem. We assume that the distribution of the fitness values of the individuals is a normal distribution, where μ and σ are the mean and the standard deviation of fitness values of individuals, respectively. Then, we regard an individual as an elite if an individual has a fitness value equal to or greater than $\mu + \alpha \times \sigma$, where α is a real number greater than 0. Then, elite degree of individual x_i^T, denoted $E(T, i)$, is defined by the following equation.

$$\begin{aligned} Elite_i^T(j) \ = \ & \{x_k^{T-j} | x_k^{T-j} \in Anc_i^T(j), \\ & \mu_{T-j} + \alpha \times \sigma_{T-j} \leq f(x_k^{T-j})\} \end{aligned}$$

$$E(T,i) = \frac{\sum_{j=0}^{level\_max} \left\{ \left| Elite_i^T(j) \right| \times \beta^j \right\}}{\sum_{j=0}^{level\_max} \left\{ |Anc_i^T(j)| \times \beta^j \right\}}$$

where $Elite_i^T(j)$ is the set of elite ancestors of x_i^T in generation $T - j$, $f(x_k^{T-j})$ is the fitness value of k-th individual in generation $T - j$. We call α the *elite decision factor* and β $(0 \leq \beta \leq 1)$ the *elite influence factor*. If α is large, the number of individuals regarded as elites decreases. Conversely, if α is small, the number of elite individuals increases. And, as β becomes small, the influence of ancestors far from the current generation decreases. For the minimization problem, we can define elite degree in a similar manner.

Using the elite degree, GA operators and GA parameters are selected during the GA execution. For example, in the crossover phase of two individuals, u and v, first the elite degrees of them, denoted $E(u)$ and $E(v)$, are calculated. Then, if $E(u) + E(v)$ is larger than the user-defined parameter, denoted D_{th}, then a crossover operator suitable for elite individuals is selected and applied. Otherwise, an operator suitable for non-elite individuals is selected and applied. The same mechanism can be used for other GA operators and GA parameters.

A Genetic Algorithm for Traveling Salesman Problems

Huai-Kuang Tsai, Jinn-Moon Yang, and Cheng-Yan Kao

Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
d7526010@csie.ntu.edu.tw

Abstract

This paper proposes an evolutionary approach for the traveling salesman problem. The proposed approach consists of global and local strategies by incorporating the family competition into edge assembly crossover and near 2-opt mutation. The method is applied to six well-known problems, including the eil101, lin318, pcb442, att532, rat575, and u724. The experimental results indicate that the proposed approach performs robustly and is very competitive with the other approaches surveyed in this paper. As our approach, although somewhat slower, executes 50 independent runs for the att532 problem, it is able to find the optimum solution in 23 runs and the average value of solution quality is 27691.3 To the best of our knowledge, the solution quality is the best in our surveys for this problem.

1 INTRODUCTION

The traveling salesman problem (TSP) is a well-known NP-hard optimization problem, in which we require to determine the shortest closed route passing through a set of *n* cities under the condition that each city is visited exactly once. Many problems in science, engineering, and bioinformatics fields, such as expert systems, routing as well as scheduling problems, flexible manufacturing systems, physical mapping problems [Alizadeh et al., 1993], and phylogenetic tree construction [Korostensky, 2000] can be formulated as a TSP. Up to now, there is no known algorithm which can determine an optimum path for the TSP bounded by any power of *n*.

A large number of approaches have been devoted to solve the TSP. Many traditional local and heuristic methods of operational research have been proposed, such as 2-opt as well as 3-opt [Lin, 1965] and Lin-Kernigan [Lin and Kernighan, 1973]. Although local search is very efficient, it often gets struck at local minima. Simulated annealing [Kirkpatrick, 1983] based on Metropolis procedure and Hopfield neural networks [Hopfield et al., 1985] have also been applied to TSP.

Evolutionary algorithms [Freisleben and Merz, 1996] [Dorigo and Gambardella, 1997] [Nagata and Kobayashi, 1997] [Tao and Michalewicz, 1998] [Mulhem and Maghrabi, 1998] [Zhenya et al., 1999] had been applied to solve TSP. An evolutionary algorithm is based on the ideas borrowed from genetics and natural selection. It is a generally adaptable concept for problem solving, especially well suited for solving difficult optimization problems, where traditional optimization methods are less efficient. There are about three main independently developed but strongly related implementations of evolutionary algorithms: genetic algorithms [Goldberg, 1989], evolution strategies [Baeck, 1996], and evolutionary programming [Fogel, 1995]. These three types of standard evolutionary algorithms are not very efficient. Thus, many modifications [Xiao et al, 1997] [Hart, 1994] [Saravanan and Fogel, 1997] [Back et al., 1997] have been proposed to improve solution quality and to speed up convergence.

Because original evolutionary algorithms are not very efficient, a trend [Hart, 1994], [Yen et al., 1998] is to incorporate local search techniques into evolutionary algorithm. Such a hybrid approach may possess both the global optimality of the genetic algorithms and also the convergence of the local searches. Another technique is to use multiple genetic operators [Saravanan et al., 1997], [Xiao et al., 1997]. The main disadvantage of this technique is that the mechanism for selecting applied operators may mislead evolutionary algorithms toward local optima.

To further improve the above approaches, in this paper a new method called family competition genetic algorithm (FCGA) is proposed for solving TSP. The approach combines the family competition, near 2-opt, and edge assembly crossover [Nagata and Kobayashi, 1997], which is an efficient genetic crossover operator. The concepts of the family competion have been successfully applied to several continuous parameter optimization problems, such as protein docking [Yang and Kao, 2000] and thin-film coatings [Yang and Kao, 2001]. The near 2-opt is a new mutation operator derived from the original 2-opt operator. In this paper, the near 2-opt is viewed as local search strategies, while the family competition and the edge assembly crossover (EAX) are viewed as global search strategies and the former is used to keep the diversity of the population. The proposed approach seems to be able to balance exploration and exploitation.

The rest of this paper is organized as follows. Section 2 introduces the evolutionary nature of the proposed approach. Section 3 provides the experimental results on

three benchmark problems and comparisons of our approach with well-known approaches, such as evolutionary approaches and local search approaches. Some characteristics of our approach are also discussed. Concluding comments are drawn in Section 4.

1. Set g =1, randomly generates an initial population, $P(g)$, with N solutions. Each solution is represented as $i = \pi(1,2,..,M)$, where π is the permutation operator, and M is the number of cities. Denote L as the family competition length.
2. Evaluate the fitness of each solution in the population $P(g)$.
3. **repeat**
   - 3.1 Let C be an empty set ($C = \phi$).
   - 3.2 **for** each solution a, called *family father*, in the population
     - **for** $l = 1$ to L
       Randomly select b from the current population
       Generate an offspring c^l by using EAX with parents a and b. (Figure 2)
       endfor
     - Select the one (c^{best}) with the lowest objective value from $c^1,\ldots,c^L$, and a. (*family selection*)
     - Generate an offspring c^m by applying *near 2-opt* operator to c^{best}. c^{best} is set to the one with better solution from c^{best} and c^m .
     - Add the c^{best} into the set C.

     endfor
   - *3.3* Let $g = g + 1$ and $P(g) = C$

   until (termination criteria are met)
   Output the best solution and the value of merit function.

Figure 1. The outline of FCGA

2 APPROACH

In this section, we present the details of our proposed algorithm for TSP. The basic ideas of FCGA are to keep the diversity of the population by the family competition and to design the genetic operators that are able to compensate for the advantages of each other. FCGA is a multi-operator approach, combining EAX and near 2-opt, that seem to be able to balance the search power of exploration and exploitation.

Figure 1 shows the main steps of FCGA. N solutions are generated as the initial population. Each solution is represented as a permutation from 1 to M, where M is the number of cities. After evaluating each individual, FCGA enters the main loop, which consists of EAX, the family competition, and near 2-opt. Each individual in the population sequentially becomes the "family father" to produce L offspring by conducting the EAX and the family competition. The near 2-opt is applied to the one, with lowest fitness value from these L offspring and the "family father", to generate a solution which is an individual of next generation. Therefore, LN solutions are generated in one generation and N solutions are selected as the parent population of the next generation.

In the following subsections, we will describe the important components including the selection mechanism, the EAX, family competition, and near 2-opt.

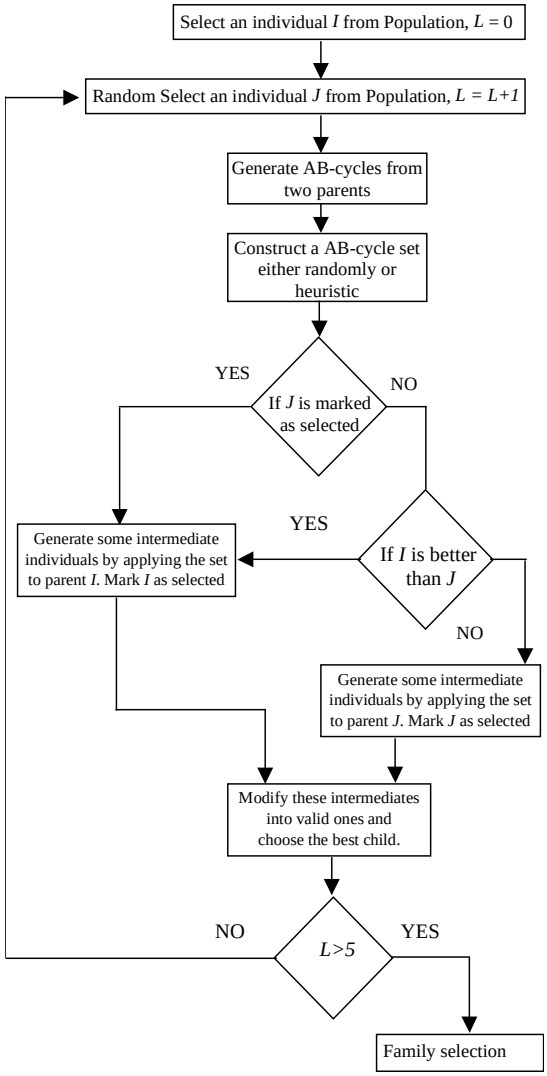


Figure 2. The outline of EAX, the selection mechanism, and family computation

2.1 SELECTION MECHANISM

A new selection mechanism derived from [D.J. Cavicchio, Jr., 1970] is proposed to integrate the search ability of the family competition and EAX. This mechanism is used to maximize the EAX searching ability and to avoid trapping to the local minima. In the

procedure of integrating the family competition and EAX (Figure 2), each individual in the population sequentially becomes the "family father." This "family father" and another solution randomly chosen from the rest of the parent population are used as parents for EAX, which is not a symmetric operator. One parent is the major parent while the other parent, "assisting father", is responsible for supplying diverse edges. If the "family father" was always the main parent during EAX, the power of exploring search space of EAX will lose. But if the parent with lower fitness value was always selected as the major parent, FCGA may easily trap to the local minima. Here FCGA maintains a vector $V = (v_1 \ldots v_n)$ to record the status whether or not each individual has been selected as the major parent. If v_i is true, it means that individual *i* had been selected as the major parent in one generation. In the selection procedure, when a parent *I* in terms becomes the "family father" to generate a child; another individual *J* is randomly selected. If v_j is true, individual *I* will be the major parent to generate children; otherwise check the fitness value of individual *J and I*. The one with lower fitness value will become the major parent to generating children. This selection mechanism not only keeps the population diversity but also preserves the search power of EAX.

2.2 EAX

Edge assembly crossover is a powerful crossover operator proposed by Nagata [Nagata and Kobayashi, 1997]. EAX has two important features. First, EAX is excellent in preserving parent's structure because most of the edges of the offspring are made from parents. Second, EAX is able to generate a wide variety of children by exploring the search space. Here EAX is biased to a global search operator in FCGA.

The EAX operator merges two parents into a single graph denoted by *G*. Two parents are denoted *A* and *B*, respectively. EAX uses the following steps to generate a new child (see Figure 2):

(1). Generate *AB-cycles*.

(2). Select "hopeful" AB-cycles.

(3). Generate intermediate individual by applying "hopeful" AB-cycles to the parent with lower fitness value.

(4). Modify the intermediate individual into a valid one.

In graph G, we trace the edge in tour-A and the edge in tour-B alternatively until we generate a cycle, and such a cycle is defined as an *AB-cycle*. Once an AB-cycle has been found, it is stored and the edges of AB-cycle are removed from *G*. This procedure is repeated until G contains no more edges. Each edge in G belongs to a unique AB-cycle. After AB-cycles have been generated, a subset of AB-cycles is chosen randomly or heuristic. Nagata [Nagata and Kobayashi, 1997] proposed a heuristic method, which uses some statistical information about two individuals versus the whole populations. The information is intuitively concerned about the gains and diversities of modified individual after applying this AB-cycle. the parameter α is set to 1 to balance these two terms.

Once the subset of AB-cycles is determined, it is applied on the parent *A* (the one with lower fitness value, denoted as the "major" parent) to generate an intermediate individual *I*. At the beginning, *I* is a duplicate of *A*. If the edge exists in parent *A*, the edge is deleted from *I*. Otherwise the edge is added to *I*. After this procedure, *I* will contain some disjoint sub-tours. Finally, EAX modifies the intermediate individual *I* into a valid child. Readers may get the more details in [Nagata and Kobayashi, 1997].

In the process of generating a child, Nagata first apply their heuristic method to generate a child. If the child is not proper to survive, then AB-cycles are randomly selected to generate another child. This random process is repeated for one hundred times. In our test and experimental results, it is proper to apply random search to find a child within the limit of 30 trails in FCGA for both time and solution concerns.

2.3 FAMILY COMPETITION

In the family competition step, EAX crossover operator creates *L* offspring and only one with best objective fitness value is survived. The procedure of the family competition is described as follows. Each individual (*a*) sequentially becomes the "family father." This "family father" and another solution (*b*) randomly chosen from the rest of the parent population are used as parents to do EAX crossover operation to generate an offspring (c^l). For each family father, such a procedure is repeated *L* times. Finally *L* solutions ($c^1,\ldots,c^L$) are produced. After *L* solutions compete with "family father," only the one (c^{best}) with the best objective value survives. Since we create *L* solutions from the same "family father" and perform a selection, this is a family competition strategy. Because each individual sequentially becomes the "family father", *LN* offspring are generated in one generation.

2.4 NEAR 2-OPT

In this subsection a new local search operator, *near 2-opt* inversion genetic operator, is proposed. Near 2-opt is based on simple inversion. It is a unary operator, since the inversion is applied to a segment of a single individual. Near 2-opt operator is adapted here to enhance the local search ability, speedup the convergence rate, and help EAX to escape from local optimum. Since EAX is a crossover operator, most edges of children are inherited from parents. Due to this property, EAX may suffer from slower convergence rate and poorer ability for exploring new edges. In other words, this operator used here is to

compensate EAX both speed and ability of searching problem space.

Figure 3 provides a description of the whole algorithm of the proposed operator. By the input of an individual *b* and *maximum inversion counter*, *MIC*, near 2-opt operator first sets the temporal individual *b'* as *b*, reset the *failed inversion counter t* as *0*, and then randomly selects a city *c* from current individual *b*. The other city *c'* is then selected by choosing the city with lowest distance to the city *c*. The section from the next city of city *c* to the city *c'* in *b'* is then inverted if *c* and *c'* are not neighbors in *b'*. If c and c' are already neighbors in b', the failed inversion t is added by 1. Selecting another city *c* and repeating above steps until the failed inversion counter *t* exceeds *MIC*. Finally, if the fitness value of *b'* is better than *b*, we replace the individual *b* by *b'*.

Let's illustrate the behaviors of near 2-opt in the following example. Assume that the current individual *b* is *(1,2,3,4,5,6,7)* and the current city *c* is *2*. If city *5* is the city with lowest distance of city *2*, the offspring *b'* will be *(1,2,5,4,3,6,7)* after we make *2-5* inversion. Now if city *2* is randomly selected again as *c*. Since the city with lowest distance is *5* and these two city are neighbors in individual *b'*, no inversion is applied and just add the failed inversion counter *t* by *1*.

1. Set the value *MIC* and the operating individual *b*, let *b'* = *b*,
2. randomly select a city *c* from *b'*, set t = 0
3. **repeat**
   - 3.1 select the city *c'* , c' is the city with lowest distance of the city *c*
   - 3.2 **if** (the city *c'* is already aligned *"next"* to the city c in the individual *b'*)
     - t = t +1
     - **if** (*t* > *MIC*) **exit** from **repeat** loop
   - 3.3 **else** inverse the section from the next city of city *c* to the city *c'* in *b'*
4. randomly select a city *c* from *b'*
5. **until** (*t*>*MIC*)
6. *if* the fitness value of *b'* is better than *b*, let *b* = *b'*
7. **return** *b*

Figure 3. The outline of near 2-opt operator

3 EXPERIMENTAL RESULTS

FCGA was implemented in C++ on a Pentium III 600MHz personal computer with single processor. Detailed information about the development of the solution qualities in each of the experiments conducted is given elsewhere [Dorigo and Gambardella, 1997] [Freisleben and Merz, 1996] [Jung and Moon, 2000]. For each test problem, fifty trials have been executed. FCGA has tested on six TSPLIB [Reinelt, 1991] benchmark problems, eil101, lin318, pcb442, att532, rat575, and u724. These problems are selected because they are widely used to compare the performances among algorithms.

Setting the values of parameters in FCGA, we have tested various values of these parameters, including the family competition length (*L*), the bias (α) and redo counter *r* in EAX, *MIC* in near 2-opt, and the population size. According to the experiments, *L* = 5 is good for both time and solution quality. The population size is the tradeoff between solution quality and convergence time. There are no obvious impacts for the others parameters. Therefore, *L = 5*, α *= 1*, r =30, and *MIC = 1* in this paper. The population size is set to the similar value of the number of cities in TSP. That is, the population sizes are set to *100*, *300*, 400, 500, 550, and *700* for eil101, lin318, pcb442, att532, rat575, and u724, respectively.

Table 1 shows the experimental results of FCGA and EGA named in [Nagata and Kobayashi, 1997] on three problems. Each problem was tested 50 independent runs. We implemented EGA according to the original paper [Nagata and Kobayashi, 1997]. "Average tour length" is the average value of the best tours found by FCEA and EGA on 50 runs as well as "Average time" and "Average generation" are values of the average time and average generation required for finding the best tours, respectively. The column, "Optimal/times" denotes the number of finding the optimal solution in 50 trails. FCGA can find the optimum solutions for eil101 and lin318 in each independent run. For the att532, FCGA is able to find the optimum solution in 23 runs and the average value of solution quality is 27691.3. FCGA, integrating the EAX and near 2-opt operators, is much better than EGA which uses only EAX. The results of EGA in this paper are better than the results in the original papers. Based on the experimental results, we find that FCGA integrate the benefits of two operators. Take the att532 case for an example, the solution quality and optimal trails are greatly improved by combining two operators. Although FCGA is somehow slower, these results show that FCGA is more robust than EGA for these testing TSP.

Table 2 shows the comparisons of FCGA with NGA [Juna and Moon, 2000], ACS [Dorigo and Gambardella, 1997], and STSP-GA [Freisleben and Merz, 1996] on lin318 and att532 problems. NGA, ACS, and STSP-GA had excellent results according to our surveys. NGA integrates NX (nature crossover) and LK local search [Lin and Kernighan, 1973]; ACS is an ant colony system with 3-opt operator; STSP-GA is the genetic algorithm with DPX [Boese, 1995] and 3-opt. The results of these three methods directly summarized from [Juna and Moon, 2000], [Dorigo and Gambardella, 1997], and [Freisleben and Merz, 1996]. "Average generation" is the values of

the average generations required for finding the best tours by the comparative methods, respectively. Table 2 shows that FCGA has better solutions than the others on these problems. In the comparison of FCGA and NGA, since NGA uses the well-known LK local search operator with smaller size of population, the time is short for solving problems. But FCGA is more stable than NGA in the solution quality without LK.

Figure 4 shows the convergence rate of the experimental results, which FCGA tested on lin318 and att532. The sketches reveal that the convergent speed of FCGA is rapid during the early search time. The speed is slow while FCGA approaches convergent state. Fortunately, FCGA often can escape from the local optimum to get the global minimum tour length.

Table 1.

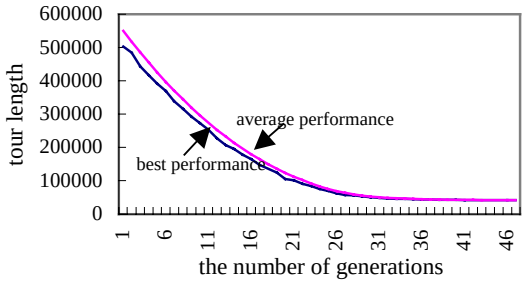
Comparisons between EGA [Nagata and Kobayashi, 1997] and FCGA on TSP problems taken from TSPLIB [Reinelt, 1991] based on the average tour length, the average time and average generation in 50 trails.

		Average time	Average tour length	Average generation	Optimal times
Eil101	FCGA	2	629	15	50
	EGA	1	629	24	50
Lin318	FCGA	60	42029	49	50
	EGA	34	42034.3	60	42
Pcb442	FCGA	232.9	50778.0	39.2	50
	EGA	116.3	50778.0	46.7	50
Att532	FCGA	304	27691.3	66	23
	EGA	226	27697.5	96	8
Rat575	FCGA	500.0	6773.2	54.8	43
	EGA	249.2	70.3	6773.333 333	35
U724	FCGA	845.1	41912.3	49.8	41
	EGA	396.7	41912.9	76.4	34

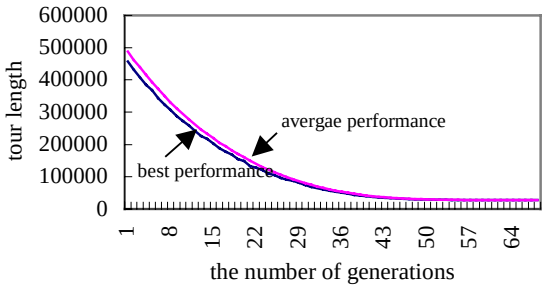
Table 2.

Comparisons between FCGA, NGA [Nagata and Kobayashi, 1997], ACS [Dorigo and Gambardella, 1997], and STSP [Freisleben and Merz, 1996] on several TSP problems taken from TSPLIB [Reinelt, 1991] based on average tour length of the best tour

	318	532
	Average tour Length	Average tour length
FCGA	42029	27691.3
NGA	42029	27695.6
ACS	42029	27718.2
STSP	42029	27693.7



(a) Tour length vs. generations: The experimental results of FCGA tested on lin318 problem.



(b) Tour length vs. generations: The experimental result of FCGA tested on att532 problem.

Figure 4. The convergence of FCGA for the lin318 and att532 problems.

4 CONCLUSION

This study demonstrates that FCGA is a stable approach for TSP. From our experience, it suggested that a global optimization method for NP-hard optimization problems should consist of both global and local search strategies. In our approach, the edge assembly crossover is a global search strategy as well as the family competition and the select mechanisms are local search strategies. These strategies can closely cooperate with each other to improve the overall search performance.

Experiments on these three test cases verify that the proposed approach is able to generate efficient solutions for TSP. Our approach can find out the optimum solution in 23 runs and the average value of solution quality is 27691.3 when it was applied the att532 problem with 50 independent runs. We believe that the flexibility and robustness of our approach make it an effective tool for TSP.

We will continue to design a more powerful operator to improve the performance as well as to speed up the convergence and will study a more diverse set of TSP to determine the limits of our approach.

References

Alizadeh, F., Karp, R.M., Newberg, L.A., and Weisser, D.K. (1993) "Physical mapping of chromosomes: a combinatorial problem in molecular biology." In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms* (SODA).

Back, T., Hammel, U., and Schwefel, H. -P. (1997) "Evolution computation: Comments on the history and current states" *IEEE. Transactions on Evolutionary Computation*.

Baeck, T. (1996) *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, USA.

Boese, K. (1995) "Cost versus Distance in the traveling salesman problem." Tech. Rep. TR-950018, UCLA CS Department.

Dorigo, M., and Gambardella, L. M. (1997) "Ant colony system: A cooperative learning approach to the traveling salesman problem." *IEEE. Transaction on Evolutionary Computation*, vol.1, no.1. pp53-66.

D.J. Cavicchio, Jr. (1970), "Adaptive Search Using Simulated Evolution." Ph.D. dissertation, Univ. of Michigan.

Freisleben, B. and Merz, P. (1996) "New genetic local search operators for the traveling salesman problem." *In Proc. PPSN IV-4th Int. Conf Parallel Problem Solving from Nature.* Berlin, Germany: Springer-Verlag, pp. 890-899.

Fogel, D. B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligent.* NJ:IEEE Press, Piscataway.

Goldberg, D. E. (1989) *Genetic algorithms in search, optimization & machine learning.* Reading. MA: Addison-Wesley.

Hart, W. E. (1994) *Adaptive global optimization with local search.* PhD thesis, University of California, San Diego.

Hopfield, J. J. and Tank, D. W. (1985) "Neural computation of decisions in optimization problems. " *Biological Cybernetics*, vol. 52, pp. 141-152.

Jung, S. and Moon B. R. (2000) "The nature crossover for the 2D Euclidean TSP" *A recombination of the 5th Annual Genetic Programming Conference (GP-2000) and the International Conference on Genetic Algorithms (ICGA-2000), Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1003-1010

Korostensky, C. and Gonnet, G. H. (2000) "Using traveling salesman problem algorithms for evolutionary tree construction." Bioinformatics Vol. 16 no. 7, pp. 619-627.

KirKpatrick, S., Gelatt C. D., and Vecchi M. P. (1983) "Optimization by simulated annealing." *Sci*, vol. 200, pp. 671-680.

Lin, S. (1965) "Computer solutions of the traveling salesman problem." *Bell Syst., J.*, vol. 23, pp. 2245-2269.

Lin, S., and Kernighan B. (1973) "An effective heuristic algorithms for the traveling salesman problem." *Operations Research* Vol.21, pp.498-516.

Mulhem, M., and Maghrabi, T. (1998) "Efficient convex-elastic net algorithm to solve the Euclidean traveling salesman problem." *Systems, Man and Cybernetics, Part B, IEEE Transactions*, Vol.284, pp.618 –620.

Nagata, Y., Kobayashi. S. (1997) "Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem." *In T. Back, editor, Proc. of the 7th Int'l. Conf. on GAs*, pp. 450-457. Morgan Kaufmann.

Reinelt, G. (1991) "TSPLIB- A Traveling salesman problem library." *ORSA Journal on Computing*, Vol.3, No.4, pp.376-384.

Saravanan, N. and Fogel, D. B. (1997) "Multi-operator evolutionary programming: A preliminary on function optimization." *In P. J. Angeline et al. editor, Proc. 6th Annu. Conf. On Evolutionary Programming (Lecture Notes in Computer Science, vol. 1213)*, pp.215-222.

Tao, G., and Michalewicz, Z. (1998) "Inver-over Operator for the TSP. " *Proceedings of the 5th Parallel Problem Solving from Nature, September 27-30, 1998, Lecture Notes in Computer Science*, pp.803-812.

Xiao, J., Michalewicz, Z., Zhang, L., and Trojanowski, K. (1997) "Adaptive evolutionary planner/navigator for mobile robots." *IEEE Trans. On Evolutionary Computation*, 1(1): pp.18-28.

Yang, J. M. and Kao, C. Y. (2000) “Flexible Ligand Docking Using a Robust Evolutionary Algorithm”, *Journal of Computational Chemistry*, vol. 21, no. 11, pp. 988-998.

Yang, J. M. and Kao, C. Y. (2001) “An evolutionary algorithm for the synthesis of multilayer coatings at oblique light incidence,” to appear in *IEEE/OSA Journal of Lightwave Technology*.

Yen, J., Yip, J. C. and Pao, Y. H. (1998) “Combinatorial optimization with use of guided evolutionary simulated annealing.” *IEEE. Trans. On Systems, Mans, and Cybernetics* –Part B, 28(2): pp.173-191.

Zhenya, H., Chengjian, W., Bingyao, J., Wenjiang, P., and Luxi. Y. (1999) “A new population-based incremental learning method for the traveling salesman problem.” *Evolutionary Computation. CEC 99. Proceedings of the 1999 Congress on*,1152-1156 Vol. 2

Building Block Superiority, Multimodality and Synchronization Problems

Clarissa Van Hoyweghen
Intelligent Systems Lab
University of Antwerp
Groenenborgerlaan 171
2020 Antwerp, Belgium
hoyweghe@ruca.ua.ac.be

David E. Goldberg
Illinois Genetic Algorithms Laboratory
University of Illinois
117 Transportation Building
104 S. Mathews Av. Urbana, IL 61801
deg@illigal.ge.uiuc.edu

Bart Naudts
Intelligent Systems Lab
University of Antwerp
Groenenborgerlaan 171
2020 Antwerp, Belgium
bnaudts@ruca.ua.ac.be

Abstract

The working of a genetic algorithm is usually explained by the search for superior building blocks. Building blocks with above average fitness are combined to construct higher order building blocks. This paper shows that this mechanism is not sufficient to solve problems where multimodality is ubiquitous. For this class of problems niching becomes a necessity. The paper analyzes the Ising model as an archetypal problem where multimodality is ubiquitous and niching is essential. The analysis introduces an important difference between searching for superior building blocks and searching for non-inferior building blocks.

1 INTRODUCTION

The practice of preserving multiple solutions in GA search has a long history [2, 11, 7, 12], the usual motivation is to preserve multiple peaks of equal or unequal fitness. The question remains whether niching is necessary in certain types of problems and, if so, whether we may better characterize the problem class where such mechanisms aren't simply a convenience, but are absolutely essential to obtain high or highest quality solutions, quickly, reliably and accurately.

The purpose of this paper is to analyze the Ising model as an archetypal problem where multimodality is ubiquitous and niching is essential, and to generalize that analysis to understand a key dividing line between different types of genetic search: the difference between searching for superior building blocks of a solution versus non-inferior building blocks of different solutions.

The paper is structured as follows. First, the Ising model is described and an intuitive explanation of why the problem is difficult for a GA is given. Then, section 2 gives a short introduction to Walsh analysis and shows how they can be used to calculate a schema's fitness value. In this section we describe the Ising model as a linear combination of Walsh monomials. Section 3 introduces the notion of non-inferior building blocks and shows how those building blocks can cause a GA to get stuck in a local optimum because of a synchronization problem. The occurrence of this synchronization problem is explained in detail and niching is proposed to avoid this problem. In section 4 we experimentally verify that a niching technique, such as sharing, can help a GA to solve a multimodal problem like the Ising model, and at the end of the paper we draw some conclusions and discuss future work.

1.1 THE STANDARD ISING MODEL

The presence of symmetry and multimodality in a problem can influence the dynamics of a search algorithm. Elsewhere [13], the one-dimensional nearest-neighbor interaction functions (NNIs) are introduced to the GA community as a class of functions that are difficult to solve because of their symmetry. The spin-flip symmetry in these problems prevents a simple GA from solving the problem quickly and can cause the GA to get stuck in a local optimum.

The standard one-dimensional Ising model [10] has its origin in statistical physics. It can be written as a one-dimensional nearest-neighbor interaction function of the form

$$f : \{0,1\}^l \to \mathbb{R} : x \mapsto \sum_{i=1}^{l} \delta(x_i, x_{i+1}) \qquad (1)$$

with string length l, $x_{l+1} \equiv x_1$ and

$$\delta(x_i, x_j) = \begin{cases} 1 & \text{if } x_i = x_j; \\ -1 & \text{otherwise.} \end{cases}$$

The maximum fitness value equals l, the minimum fitness value equals $-l$ and the average fitness value equals zero.

The Ising model clearly contains spin-flip symmetry: flipping all ones in zeros and visa versa does not change the

fitness value. This characteristic implies that the problem has two maxima, the string containing only ones and the string containing only zeros. In the next section, we give an intuitive explanation of how a GA tries to solve the Ising model following the arguments presented elsewhere [13].

1.2 THE RANDOM WALKS OF DOMAINS

Consider the following bit string

$$0000|1111111|000.$$

Sequences of the same values are called *domains* and pairwise combinations of a 1 and 0 are called *domain walls*. The fitness value of the bit string is only 4 less than the fitness of the optima, but in Hamming distance the problem is far from an optimum. Mutating a bit in the interior of a domain, for example

$$0000|111|0|111|000,$$

creates two new walls and decreases the fitness value with 4. Mutating a bit next to a domain wall, for example

$$00000|111111|000,$$

moves the wall to the left or the right and leaves the fitness value unchanged. The only way to increase the fitness value is to make two domain walls collide. So solving the Ising model with a single-bit-flip hill-climber results in a sequence of random walks of domain walls.

When trying to solve the Ising model using a GA, experiments show that as the building blocks grow, diversity gets lost and the population becomes a set of strings all converging to the same optimum and with domain walls at about the same string positions. At that point, a normal crossover operator, like two-point crossover, produces offspring which are worse than their parents,

$$\left.\begin{array}{l} 000011\,|\,11\,|\,110000 \\ 000111\,|\,10\,|\,000000 \end{array}\right\} \Rightarrow \begin{array}{l} 000011\,|\,10\,|\,110000 \\ 000111\,|\,11\,|\,000000, \end{array}$$

or in the best case equally good,

$$\left.\begin{array}{l} 000011\,|\,111100\,|\,00 \\ 000111\,|\,100000\,|\,00 \end{array}\right\} \Rightarrow \begin{array}{l} 000011\,|\,100000\,|\,00 \\ 000111\,|\,111100\,|\,00. \end{array}$$

So again, solving the problem becomes a sequence of random walks of domain walls.

In the next sections, we perform a Walsh analysis of the Ising model to understand why the model is difficult to solve for a GA and show how niching can prevent a GA from getting stuck in a local optimum.

2 WALSH ANALYSIS OF THE ISING MODEL

In this section, the Ising model is described in terms of Walsh coefficients which make it easier to investigate the performance of a GA solving this problem. We first give a brief introduction to Walsh functions and show how they simplify the calculation of schema average fitness, which was first shown by Bethke [1]. The notation we will use is developed elsewhere [4] and for a more elaborate introduction to Walsh functions and their application to the analysis of a GA's performance, we refer to this work.

2.1 INTRODUCTION WALSH FUNCTIONS

A GA processes bit strings, $x = x_l \ldots x_1$, with $x_i \in \{0,1\}$ and string length l. Every bit string $x \in \{0,1\}^l$ can be mapped to an auxiliary string $y \in \{-1,1\}^l$ by using the mapping

$$y_i = 1 - 2x_i,$$

which simply states that a 0 in the bit string maps to a 1 in the auxiliary string and a 1 in the bit string maps to a -1 in the auxiliary string. Using this mapping, the Walsh functions can be defined as a set of 2^l monomials over the auxiliary string variables:

$$\varphi_j(y) = \prod_{i=1}^{l} y_i^{j_i},$$

with the index counter j used bit by bit to determine whether the ith string position is represented in the product or not. In other words, the Walsh functions can be seen as partial parity functions, which return -1 or 1 if the number of 1s in the auxiliary string y over the bit positions defined by the 1s in the binary representation of the index j is odd, respectively even. For instance, for $j = 6 = 110_2$, $\varphi_6(011) = 1^1(-1)^1(-1)^0 = -1$ and the number of ones in the two most left positions in string 011 is odd. Table 1 shows the index j, its binary representation and the corresponding Walsh monomial for $l = 3$ auxiliary strings.

j	binary j	$\varphi_j(y)$
0	000	1
1	001	y_1
2	010	y_2
3	011	y_2y_1
4	100	y_3
5	101	y_3y_1
6	110	y_3y_2
7	111	$y_3y_2y_1$

Table 1: Table of the Walsh monomials for $l = 3$.

Because the Walsh functions form an orthogonal basis over the auxiliary strings, the fitness function f can be written as a linear combination of the Walsh monomials

$$f(x) = \sum_{j=0}^{2^l-1} w_j \varphi_j(x),$$

mapping bit strings to auxiliary strings in the term $\varphi_j(x)$ when necessary.

2.2 ANALYSIS OF THE ISING MODEL IN WALSH COEFFICIENTS

The Ising model described in Eq.1 can easily be transformed to the function

$$f(y) = \sum_{i=1}^{l} y_i y_{i+1}$$

over the auxiliary string y, with $y_{l+1} \equiv y_1$. Looking at this transformation it becomes immediately clear which the Walsh coefficients of the Ising model are. The only non-zero Walsh coefficients are the second-order Walsh coefficients which have a binary representation for their index j containing all zeros except for two adjacent positions. Table 2 shows all non-zero Walsh coefficients, the corresponding binary representation for j for $l = 6$ bit string and the corresponding Walsh monomials.

$w_j = 1$	binary j	$\varphi_j(y)$
w_3	000011	$y_2 y_1$
w_6	000110	$y_3 y_2$
w_{12}	001100	$y_4 y_3$
w_{24}	011000	$y_5 y_4$
w_{48}	110000	$y_6 y_5$
w_{33}	100001	$y_6 y_1$

Table 2: Table of all non-zero Walsh coefficients of the Ising model for $l = 6$ bit strings.

Mathematically, the Walsh coefficients of the Ising model can be described by

$$w_j = \begin{cases} 1, & \text{if } j \in \{3 * 2^i : \forall 0 \leq i \leq l-2\}; \\ 1, & \text{if } j = 2^{l-1} + 1; \\ 0, & \text{otherwise.} \end{cases}$$

2.3 SCHEMA AVERAGE

Using the Walsh functions, the schema average fitness of a schema $h = h_l \ldots h_1$ can be written as a sum of Walsh coefficients:

$$f(h) = \sum_{j \in J(h)} w_j \varphi_j(\beta(h))$$

with the function β mapping all $*$s to a 0 and the sum taken over all js having a binary representation which is an element of the index set $J(h)$:

$$J(h_i) = \begin{cases} 0 & \text{if } h_i = *; \\ * & \text{if } h_i = 0, 1. \end{cases}$$

For example $J(*1*) = 0*0$ which can be seen as a schema for the index set $\{000_2, 010_2\} = \{0, 2\}$ and $J(00*) = **0$ which can be seen as a schema for the index set $\{000_2, 010_2, 100_2, 110_2\} = \{0, 2, 4, 6\}$. Table 3 gives the fitness average of some $l = 3$ schemata expressed in Walsh coefficients. Notice that low-order schemata, which are more general schemata, are specified by a short sum of Walsh coefficients and high-order schemata, which are more specified schemata, are specified by a long sum.

Schema	Fitness average
$***$	w_0
$**0$	$w_0 + w_1$
$**1$	$w_0 - w_1$
$*0*$	$w_0 + w_2$
$1**$	$w_0 - w_4$
$*11$	$w_0 - w_1 - w_2 + w_3$
$1*0$	$w_0 + w_1 - w_4 - w_5$
111	$w_0 - w_1 - w_2 + w_3 - w_4 + w_5 + w_6 - w_7$

Table 3: Some $l = 3$ schemata and their fitness average expressed in Walsh coefficients.

3 BB-SUPERIORITY VERSUS NON-INFERIORITY

In this section the notions of non-inferior BBs and synchronization problems are introduced. We will see that the Ising model contains many non-inferior BBs and that their existence can cause a GA to get stuck in a local optimum because of a synchronization problem. Niching will be proposed to avoid these synchronization problems. At the end of the section we introduce some badly scaled Ising models which act like needle-in-a-haystack problems for a GA without niching.

3.1 SYNCHRONIZATION PROBLEMS

The working of a GA can be explained by use of the *building block* hypothesis [8, 5]. GAs process minimal sequential superior building blocks (BBs) in such a way that low-order BB-blocks with above average fitness are combined to construct higher-order BB-blocks. The schema theorem [8] gives a lower bound for the expected number of individuals containing a schema h at a particular time t:

$$m(h, t+1) \geq m(h, t) \frac{f(h)}{\overline{f}} \left[\; 1 - p_c \frac{\delta(h)}{l-1} - p_m o(h) \; \right]$$

with $m(h,t)$ the expected number of BBs at time t, $\overline{f}$ the average fitness value, p_c the crossover rate, p_m the mutation rate, $\delta(h)$ the defining length of the schema and $o(h)$ the order of the schema.

The Ising model contains only a small portion of non-zero Walsh coefficients. The zeroth-order coefficients tells us that the average of the fitness function is zero, $f(******) = w_0 = 0$. Because all first-order coefficients are also zero, it is easy to see that the first-order BBs,

$$\begin{aligned} f(*****0) &= w_0 + w_1 = 0, \\ f(****1*) &= w_0 - w_2 = 0, \\ f(***0**) &= w_0 + w_4 = 0, \\ f(**1***) &= w_0 - w_8 = 0, \end{aligned}$$

do not teach us anything about the optima of the problem. Investigating the second-order we notice that l of the $\binom{l}{2}$ coefficients have the value 1. Table 2 shows that these coefficients favor schemata having the same value at two adjacent string positions.

$$\begin{aligned} f(****11) &= \psi_0 - \psi_1 - \psi_2 + w_3 = 1 = f(****00) \\ f(****10) &= \psi_0 + \psi_1 + \psi_2 - w_3 = -1 = f(****01) \end{aligned}$$

Because all Walsh coefficients of order higher than two are zero, the fitness average of all BBs of order higher than two is exactly determined by the non-zero second-order coefficients. This causes that at any order q, all BBs having q adjacent fixed positions all set to the same value have the same fitness value and their fitness value is superior to all other order-q BBs:

$$\begin{aligned} f(***111) &= f(***000) = f(**111*) > f(**100**) \\ f(*1111*) &= f(*0000*) = f(111**1) > f(**1001*) \end{aligned}$$

BBs which are equally good and superior to all alternatives are called *non-inferior* BBs [6].

Following the Schema Theorem, non-inferior BBs, like $***111$ and $***000$, should grow with the same proportion. Unfortunately, this does not always happen due to genetic drift and the limited size of the population. It is even possible that some non-inferior BBs disappear out of the population. If at that moment for example all BBs in the population which fix more string positions to the right are BBs of optimum 000000 and all BBs in the population which fix more string positions to the left are BBs of optimum 111111, the BBs cannot be combined to form higher-order BBs of an optimum. We say that the GA has a *synchronization problem* [14]: it is stuck in a local optimum because the schemata in its current population, which are non-inferior BB of different optima, cannot be combined. Following example describes a synchronization problem in more detail. Suppose the population contains only BBs $11****$ and $****00$; their non-inferior counter parts $00****$ and $****11$ are not present in the population. The combination of the BBs in the population, $11**00$, is not a BB of an optimum and does not lead the GA to an optimum. The GA has a synchronization problem. Solving the problem is left to the mutation operator whose working can be compared with a sequence of random walks of domain walls.

Why is the probability that a synchronization problem occurs so high when solving an Ising model with a GA? This question can easily be answered by investigating BBs from which the fixed positions are not all adjacent. BBs like $*11*00*$ which are not BBs of a solution and BBs like $*00*00*$ which certainly are BBs of an optimum have the same average fitness value because of the few non-zero Walsh coefficients. This means that they both have the same chance to be selected for reproduction, although one of them is not a BB block of an optimum and leads the GA to a synchronization problem when losing diversity. If the problem would have more non-zero Walsh coefficients, as is the case when all second-order coefficients are set to 1, BBs having the same value at all their fixed positions would have an higher average fitness value than other BBs. The GA would decide faster to which of the two optima it would converge and when diversity gets lost, the population will still contain enough BBs of the same optimum to find it.

3.2 SELECTION PRESSURE WITH NICHING

The key problem of a GA stuck in a local optimum due to a synchronization problem is the existence of non-inferior BBs of different solutions. Due to genetic drift and the limited size of the population it is possible that such non-inferior BBs do not grow in the same proportion and that some of them eventually disappear. In case of the Ising model, this results in a population which is not diverse enough to contain the necessary BBs to construct an optimum.

A well-known technique to maintain diversity in the population is niching. By dividing the population in different niches non-inferior BBs will grow in the same proportion and the GA will process more in terms of minimal non-inferior sequential BBs instead of minimal sequential superior BBs [6].

The Ising model shows that for a certain class of optimization problem niching becomes a necessity for a GA to solve these problems. This statement is even stronger than in [9] where it is stated that even without the need of a diverse population, for example to obtain multiple optima, niching can be beneficial and enhance the performance of a GA.

The niching technique used in this paper is sharing [7]. Fitness sharing accomplishes niches by degrading an individuals fitness according to the similarity with the other individuals in the population. The sharing function Sh is

defined as follows

$$Sh(d_{i,j}) = \begin{cases} 1 - \left(\frac{d_{i,j}}{\sigma_{sh}}\right)^{\alpha_{sh}} & \text{if } d_{i,j} < \sigma_{sh}; \\ 0 & \text{otherwise;} \end{cases}$$

with $d_{i,j}$ the hamming distance between individuals i and j. The parameter σ_{sh} determines the proximity of the individuals with who an individual has to share. In [3], Deb and Goldberg proved that we may approximate this parameter by using

$$\sigma_{sh} = \frac{1}{2}(l + \frac{1}{q}\sqrt{l}),$$

with q the problem's number of optima. The other parameter α_{sh} is mostly set to one, giving us the *triangular sharing function*. For each individual i the niche count m_i is calculated as

$$m_i = \sum_{j=1}^{n} Sh(d_{i,j})$$

with n the population size. The *shared fitness* f_{sh} of an individual i is then

$$f_{sh}(i) = \frac{f(i)}{m_i}.$$

3.3 SCALING

Before experimentally verifying our theory which states that niching is necessary to solve the Ising model, we introduce the scaling aspect in the Ising model. This can make the Ising model even more difficult for a GA.

When not all non-zero Walsh coefficients are set to one, but some of them have an higher value, a phenomenon called *pinning* [13] can occur. When an Ising model is not well scaled, the GA will pay more attention to BBs with high Walsh coefficients because of their high fitness contribution and the processing of the other BBs will be delayed. If at a moment diversity gets lost and some of the non-inferior BBs disappear a synchronization problem can occur. The processing of the Ising model can then again be compared with a sequence of random walks of domain walls. Domain walls which where first free to walk until they met another wall and collide, can now only walk in a restricted area because of the BBs with high Walsh coefficients. For example in

$$\underbrace{000000}_{\text{h}}|\underbrace{1111}_{\text{h}}|0000$$

($\underbrace{\smile}_{\text{h}}$ is a high BB) a domain wall will never break up a high BB because this will decrease the fitness value of the string too much. A random walk can only solve the problem if all BBs with high Walsh coefficients are BBs from the same optimum.

Elsewhere [13], Naudts and Naudts construct some badly scaled Ising models which are needle-in-a-haystack problems for a GA. Experiments in this work show that those Ising models cannot be solved by a GA within 30000 iterations. In next section we will show that a GA with niching can solve badly scaled Ising models.

4 EXPERIMENTAL VERIFICATION

In this section we verify experimentally that niching is necessary for a GA to solve the Ising model accurately. The algorithm used in the experiments is a simple genetic algorithm with two-point crossover, rate 0.8. The mutation rate is set to zero to investigate the influence of the recombination operator. As niching technique, sharing with σ_{sh} set to $\frac{1}{2}(l + \frac{1}{q}\sqrt{l})$ is used. Experiments show that the value of the parameter q is not so important when solving the Ising model.

An important issue when solving an Ising model is the diversity of the population. Therefore, we define a diversity measure which gives the percentage of ones at a certain string position for all strings in the population. For example, a diversity of 1 (respectively 0) for string position i means that all strings have value one (respectively value zero) at that string position. A diversity of p, with p between 0 and 1, indicates that p percentage of the population has value one at that string position and $1 - p$ percentage has value zero. The population is perfectly diverse if the diversity measures for all string positions are 0.5.

Without niching a GA gets stuck in a local optimum because of a synchronization problem. The population is not diverse enough and a solution cannot be reached by mixing the BBs in the population. Figure 1 shows a diversity plot for a GA trying to solve an $l = 80$ (well scaled) Ising model without niching. The population size is set to 1000. The figure shows that already quite early in the run all strings in the population have value one for a certain set of string positions and value zero for an other set of string positions. A synchronization problem occurs and recombination of BBs cannot help the GA to find the optimum.

By sharing an individual's fitness according to the similarity with the other individuals in the population, the algorithm tries to keep its population diverse. The size of the population is hereby an important factor. If the population size is too small, sharing cannot maintain the diversity and a synchronization problem can occur. Figure 2 shows the diversity of the population while solving the $l = 80$ Ising model with sharing. A population size of 60 is not big enough to maintain diversity. In contrast, figure 3 shows that a population size of 140 is big enough to maintain diversity and the solution is found within 68 generations.

Figure 4 shows the minimum population size that a GA

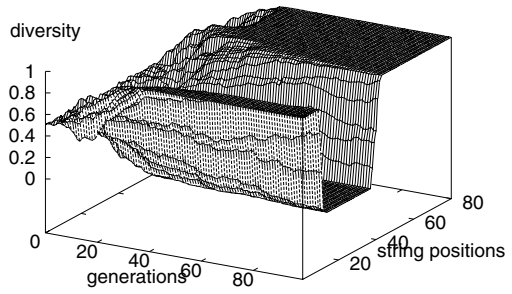


Figure 1: Representation of the diversity of the population while solving an $l = 80$ Ising model without niching.

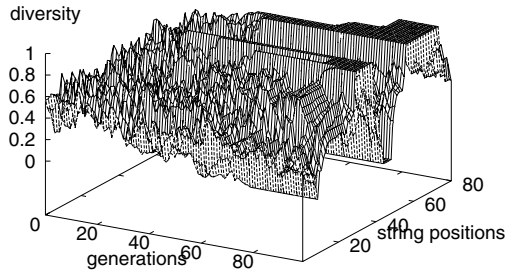


Figure 2: Representation of the diversity of the population while solving an $l = 80$ Ising model with sharing. A population size of 60 is not big enough to maintain diversity.

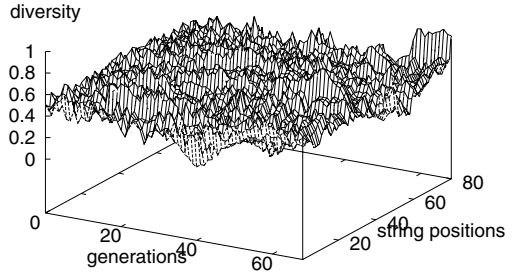


Figure 3: Representation of the diversity of the population while solving an $l = 80$ Ising model with sharing. The population size is 140 and the optimum is found after 68 generations.

with sharing needs to solve the Ising model. The parameter q which determines σ_{sh} is set to 2 and the population size n is empirically determined to be sufficiently large to find an optimum for 30 independent runs. The population size for a normally scaled Ising model scales up subquadratically (about $n = l^{0.627}$) which is acceptable and similar to other known models. Changing the parameters q does not change the minimum population size significantly. Similar results are found for badly scaled Ising models. The badly scaled Ising models used in the experiments are

$$f : \{0,1\}^l \to \mathbb{R} : x \mapsto \sum_{i=1}^{l} \delta(x_i, x_{i+1})$$

with string length l, $x_{l+1} \equiv x_1$ and

$$\delta(x_i, x_{i+1}) = \begin{cases} a & \text{if } i \bmod b = 0 \text{ and } x_i = x_{i+1}; \\ 1/2 & \text{if } i \bmod b \neq 0 \text{ and } x_i = x_{i+1}; \\ 0 & \text{otherwise.} \end{cases}$$

These models where first defined in [13] where it is shown that those badly scaled Ising models cannot be solved accurately by a GA within 30000 iterations. Figure 4 shows that a GA using sharing can solve badly scaled Ising models using reasonable small populations.

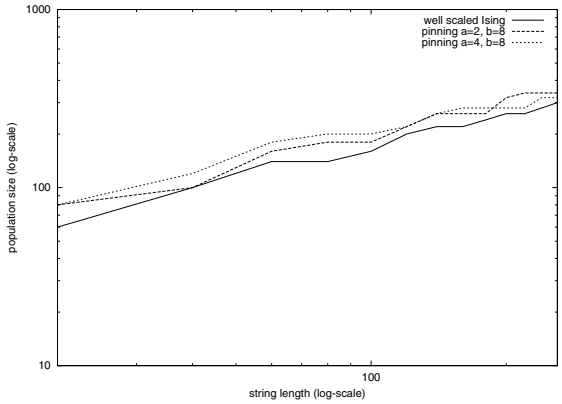


Figure 4: Minimum population the sharing GA needs to solve the Ising model. The full line represents the normally scaled Ising model, the dashed line represents the badly scaled Ising model with a set to 2 and b set to 8, the dotted line represents the badly scaled Ising model with a set to 4 and b set to 8.

The Ising model can be seen as a base model for a GA with niching. We claim that it is one of the simplest optimization problem for which niching becomes a necessity to find an optimum. Where mutation was essential to solve an Ising model without niching by a sequence of random walks of the domain walls, the crossover operator becomes essential in the search process of a GA with niching. Figure 5 shows the average number of generation a GA with sharing needs to find an optimum. The population size is set to 300, which is big enough for all string lengths between 20

and 260. The average is taken over 30 independent runs. The algorithm still uses only two-point crossover with rate 0.8 and has no mutation operator. The figure shows that by mixing good BBs an optimum can be found quickly. Adding a mutation operator with a high mutation rate, for example $\frac{1}{l}$, even prolongs the search process. Every mutation that flips a bit which is not situated next to a domain wall introduces two new domain walls. This neutralizes the positive effect of a successful recombination.

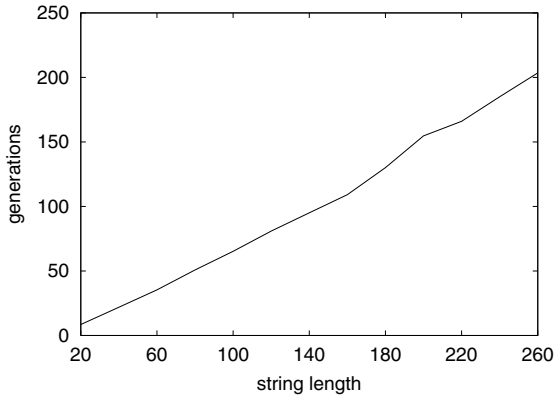


Figure 5: Average number of generations versus string length for a GA using sharing and with population size 300.

5 CONCLUSIONS AND FUTURE WORK

The paper showed that niching is not only a good technique to preserve multiple solutions in GA search, but for a certain class of problems niching becomes a necessity to solve the problem quickly, reliably and accurately. By analyzing the Ising model as an archetypal problem where multimodality is omnipresent, we tried to understand the difference between searching for superior building blocks of a solution and searching for non-inferior building blocks of different solutions. In the future we would like to better specify the class of problems where the search for non-inferior building blocks is essential. We think of clustered problems in which every cluster has multiple solutions, but not all solutions of different clusters match together. We would also consider hierarchical problems, like the H-IFF problem [15], where the choice between low level non-inferior BBs should be postponed until a higher level is reached.

Acknowledgments

The authors would like to thank Martin Pelikan for many useful discussions. The first author wishes to thank the third author for the encouragement to visit the Illinois Genetic Algorithms Laboratory (IlliGAL) as a visiting scholar. Most of the work for this paper was done during this visit.

The first author is supported by the Flemish Institute for the Encouragement of Scientific and Technological Research in Industry – (IWT) (Flanders) (Belgium). The third author is a Post Doctoral Fellow of the Fund for Scientific Research – (FWO) (Flanders) (Belgium).

The work was partly sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U.S. Army, or the U.S. Government.

References

[1] A. D. Bethke. *Genetic algorithms as function optimizers*. PhD thesis, University of Michigan, Ann Harbor, 1980. *Dissertation Abstract International*, 41(9), 3503B. (University Microfilms No 81-06101).

[2] D. J. Cavicchio. *Adaptive search using simulated evolution*. PhD thesis, University of Michigan, Ann Harbor, 1970. (University Microfilms No 25-0199.

[3] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, Inc., 1989.

[4] D. E. Goldberg. Genetic algorithms and Walsh functions: Part 1, a gentle introduction. *Complex Syxtem*, 3:129–152, 1989.

[5] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[6] D. E. Goldberg. *The design of competent genetic algorithms: Towards a computational theory of innovation*. Unpublished manuscript, 2001.

[7] D. E. Goldberg and J.Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. Grefenstatte, editor, *Proceedings of the Second International Conference on Genetic Algorithms.*, pages 41–49. Lawrence Erlbaum Associates, 1987.

[8] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.

[9] J. Horn. *The nature of niching: genetic algorithms and the evolution of optimal, cooperative populations.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, 1997. Also IlliGAL Report No. 97008.

[10] E. Ising. Beitrag zur Theorie des Ferromagnetismus. *Z. Physik*, 31:235, 1924.

[11] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975. *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No 76-9381).

[12] S. W. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, 1995. Also IlliGAL Report No. 95001.

[13] B. Naudts and J. Naudts. The effect of spin-flip symmetry on the performance of the simple GA. In A.E. Eiben, Th. Bäck, M. Schoenauer, and H.P. Schwefel, editors, *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, pages 67–76. Springer-Verlag, LNCS 1498, 1998.

[14] C. Van Hoyweghen and B. Naudts. Symmetry in the search space. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC2000)*, pages 1072–1078. IEEE Press, 2000.

[15] R. Watson, G. S. Hornby, and J. B. Pollack. Modeling building block interdependency. In A.E. Eiben, Th. Bäck, M. Schoenauer, and H.P. Schwefel, editors, *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, pages 97–106. Springer-Verlag, LNCS 1498, 1998.

Coevolutionary Dynamics in a Minimal Substrate

Richard A. Watson **Jordan B. Pollack**

Dynamical and Evolutionary Machine Organization,
Volen Center for Complex Systems, MS018, Brandeis University,
Waltham, MA 02454, USA
richardw@cs.brandeis.edu

Abstract

One of the central difficulties of coevolutionary methods arises from 'intransitive superiority' – in a two-player game, for example, the fact that *A* beats *B*, and *B* beats *C*, does not exclude the possibility that *C* beats *A*. Such cyclic superiority in a coevolutionary substrate is hypothesized to cause cycles in the dynamics of the population such that it 'chases its own tail' - traveling through some part of strategy space more than once despite apparent improvement with each step. It is often difficult to know whether an application domain contains such difficulties and to verify this hypothesis in the failure of a given coevolutionary set-up. In this paper we wish to elucidate some of the issues and concepts in an abstract domain where the dynamics of coevolution can be studied simply and directly. We define three simple 'number games' that illustrate intransitive superiority and resultant oscillatory dynamics, as well as some other relevant concepts. These include the distinction between a player's perceived performance and performance with respect to an external metric, and the significance of strategies with a multi-dimensional nature. These features alone can also cause oscillatory behavior and coevolutionary failure.

Keywords: Coevolution, intransitive superiority, multiple dimensions, coevolutionary failure.

1 INTRODUCTION

Coevolution has become increasingly popular in Evolutionary Algorithms research (e.g. Hillis 1992, Sims 1994, Juille 1996, Miller & Cliff 1994). The basic idea behind the approach seems intuitive enough – rather than evolve individuals against a fixed objective metric, we engage individuals in the task of improving their performance against other evolving individuals. One of the most unequivocal benefits of this approach comes from the fact that for many machine learning domains a suitable objective metric of performance is simply not available. Examples include the coevolution of pursuit and evasion behaviors (Miller and Cliff 1994, Reynolds 1994), and competitive manipulation of physical objects (Sims 1994). Apart from this primary benefit of providing some target for performance, coevolution is commonly understood to have several other benefits. The following list is not a comprehensive account of coevolution's supposed benefits – rather we have selected those ideas for which we will be able to illustrate related issues in our experiments – but, these ideas cover some of those most common in the coevolution literature. We use examples from the domain of chess but the concepts apply equally to any task that can be described using performance with respect to an opponent:

a) Providing a target that is 'hittable' – gradient.
If any two novice chess players play against, say, a Grand Master then both will lose and their performance will be indistinguishable. But if the novices play against each other the superiority of one with respect to the other will be revealed. By engaging players in the mutual pressure to outperform one-another coevolution provides adaptive gradients that might otherwise be hard to engineer. Pollack & Blair (1996) provide an example where 'self-play' provides a gradient for learning.

b) A target that is relevant – focusing.
If we have our two novices play against a (fixed) set of other chess players of various abilities then the number of games they win might be different, and we may select the better. But how are we to devise this set of opponents? A random set may not be representative of all chess players. Any given set may fail to test certain aspects of play. By using other evolving players as opponents coevolution may focus adaptation on those aspects of a task that have not yet been optimized. Examples where one 'species' is used to provide a focused test-set for another species include Hillis (1992), and Juille (1996).

c) A moving target – open-endedness.
Even if we could find a representative fixed set of opponents that provided a gradient from novice play through to master level, any fixed set will have an

upper limit. Using coevolution there is always the potential to be a better player than the best player found so far, and when found such a player provides the new target to beat. Open-endedness is often cited as a benefit of coevolution, e.g. (Ficici 1998)

However, although these notions are common, and seem intuitive enough, they are not very well defined. Moreover, there is increasing awareness, in this same community of researchers, that coevolution can sometimes introduce as many problems as it solves. In the list below we describe informally some of the ways in which a coevolving target for performance might not be hittable, relevant, or moving in the right direction:

a) Loss of gradient.
Suppose an evolving population of opponents becomes too good – we may find ourselves with an 'unhittable' target once more. For example, perhaps evasion is very much easier than pursuit, and none of the evaders can be caught. In this case we lose the gradient information and players may drift without improvement.

b) Focusing on the wrong things.
The ability to focus on an opponent's weakness can provide an easy way to win. This may produce degenerate players that over-specialize on opponents weaknesses, and fail to learn a task in a general way.

c) 'Relativism'.
When opponents co-adapt, and describe a task for one another, we suppose that they will 'leap-frog' one another in steps of increasing performance. But, if *A*'s performance is defined by *B*, and *B*'s performance is defined by *A*, then the adaptive system is disconnected from any absolute measure of performance. Two good players get the same score against each other, as do two bad players. So, supposing variation is equally likely to take the standard of play down as up (perhaps more likely), what is to ensure that these moving targets will move the way we want them to? Such relativism may enable ways for the players to 'subvert' the game we as researchers had in mind, and may lead to mediocre players that never improve.

Problems such as these may be involved in some of the failures in the literature, but it is very difficult to be sure. Identifying the cause of a failure is complicated by the fact that it is very difficult to separate the dynamics of the coevolutionary set-up from the details of the application domain, be it backgammon, robotics, pursuit and evasion, or whatever. Thus, the benefits and the problems with coevolution continue to be a bit vague and ill-defined; often going no further than the level of description we have given above, and relying on metaphors like 'arms race' and 'collusion'.

Some work has addressed issues in coevolution and relativism in the abstract, which enables particular underlying concepts to be illustrated and investigated (Maynard-Smith 1982, Cliff & Miller 1995, Kauffman 1993). In this style, we introduce in this paper a minimal substrate in which coevolutionary concepts, dynamics, and problems can be investigated - in particular, the importance of intransitive superiority. Specifically, we evolve scalar values and vectors under various coevolutionary set-ups. This substrate enables us to illustrate some important concepts that may be underlying the problems we introduced above. Our experiments provide concrete examples for each of the ideas we have discussed, and assist us in gaining some defining concepts that may be useful in diagnosing coevolution failures.

The following sections are organized as follows: Section 2 introduces some of the concepts we see as central to the issues we described above, and describes the minimal substrate we use for our investigations. Section 3 describes other aspects of our coevolutionary set-up. Section 4 describes experiments that each illustrate a different potential cause of failure in coevolution. Section 5 concludes.

2 A MINIMAL SUBSTRATE

In this section we introduce the minimal substrate that we will use in our experiments. In the process we will describe some of the concepts that we see as important for understanding the issues involved in coevolution.

2.1 SCALARS

We commence by considering the coevolution of scalar values. For example, we could evolve integers using coevolutionary techniques to find high values. In this domain we know that the task is trivial and that *evolving* integers is easy, thus any problems we have using *coevolutionary* techniques are a product of coevolution. Although it may seem too trivial to be of use we will see that there are several phenomena that can be illustrated with its help.

We assert that the goal of the evolutionary process is to maximize, *a*, the value being evolved. Clearly, if we evolve integer values using a fitness function, $f(a)=a$, then the problem is trivial. However, we will investigate what happens when we coevolve these values using a fitness function, $f(a,S)$, that returns a value for one number, *a*, *with respect to* a set of other numbers, *S*. *S* is a *sample* of individuals against which *a* will be tested. For the purposes of our experiments we will use $f(a,S)$ that counts the number of members of *S* that are less than *a*:

$$f(a,S) = \sum_{i=1}^{|S|} score(a,S_i) \qquad \textbf{/eq.1}$$

where $score(a,b)=1$ if $a>b$, 0 otherwise.

In this way we may evolve the scalar values as if they were playing a 'greater than' game, rather than evolving them against an objective fitness function.

Clearly, if *S* were the complete set of possible values in the domain of *a*, then our coevolutionary set-up would be

the same as the trivial evolutionary case. But naturally, for our purposes S will consist of other coevolving individuals. In some cases these will be drawn from the same population and may therefore be genetically related. In other cases, they will be drawn from a separate coevolving population. We will see that the effects of this choice can be significant.

2.2 OBJECTIVE FITNESS, AND SUBJECTIVE FITNESS

In evolutionary algorithms the fitness of an individual is given by a 'fitness function' or 'objective function' - this provides some measure of the individual's performance or quality with respect to the task at hand. In coevolution, there is still a fitness function, e.g. Equation 1 - but the value it returns is no longer objective, it is subject to the sample chosen. To make the distinction clear, we will call the metric that we as researchers seek to optimize the *objective fitness*, and we will call the metric of performance as perceived by the co-evolving individual the *subjective fitness*.[1]

We asserted above that our objective fitness was $f(a)=a$. It seems fairly likely that anything adapting under our 'greater than' game will become maximized, as we intended, but this is not necessarily so. Consider making a judgment: which of a and b is to be preferred? (where a and b are any two individuals). With respect to our objective metric, the preferred individual is whichever is larger in value. They may be equally preferred only if they are equal in value. Let us denote this objective preference relation as $P_{obj}(a,b)$. In the coevolutionary game, the one that will be preferred will be whichever gets the highest value when played in the 'greater than' game against S. This is its *subjective* fitness. If $f(a,S_a)$ is greater than $f(b,S_b)$ (according to Equation 1) then a is preferred. Let us write the coevolutionary preference as $P_{subj}(a,b)$, then we have stated that $P_{subj}(a,b)=P_{obj}(f(a,S_a), f(b,S_b))$. Notice that we do not assume that a and b are evaluated against the same S. And it should be clear that we may get a different preference depending on how we choose S_a and S_b.

Many of the problems we encounter in coevolution can be described as arising from the separation between a player's performance as they perceive it, from their performance with respect to an external metric. A mismatch of preference relations from objective and subjective metrics, i.e. $P_{subj}(a,b) \neq P_{obj}(a,b)$, will occur depending on the choice of S. Clearly if any choice of S is possible then we can reverse the preference relation of a and b. For example, suppose, a =4 and b=5, so $P_{obj}(a,b)$ returns b. If we choose S_a={1,2,3} and S_b={6,7,8} then $f(a,S_a)$=3, and $f(b,S_b)$=0, so $P_{subj}(a,b)=a$. The subjective and objective preferences give opposite answers.

Even if S is the same for both a and b we can get an erroneous result. Consider, $S_a=S_b$={1,2,3}. Both a and b score the same as each other because they win against all opponents. Alternatively, we can choose S so that they lose against all opponents. So, $P_{subj}(a,b)$="draw". This corresponds to the 'loss of gradient problem' we described in the introduction.

To be sure, in a coevolutionary set-up, the composition of S is not arbitrary. But, we must be aware that even though the choice of a coevolutionary game may not seem problematic, we have already disconnected from the objective measure of performance. We will see that even our simple 'greater than' game can cause problems even in a quite normal coevolutionary set-up. Whereas, in an applied coevolution, the absence of an objective metric can prevent us from examining what is really happening, here we are able to illustrate these concepts clearly because we have access to both the subjective and objective fitnesses.

2.3 MULTIPLE DIMENSIONS

The second feature of our minimal substrate is the introduction of additional dimensions to the definition of an individual. That is, we may represent individuals by pairs of scalars, or vectors. For simplicity, let us discuss pairs, and call the two dimensions **x** and **y**. We will let each dimension represent a different aspect of a player's abilities. It is important to realize that we cannot necessarily reduce multiple dimensions to a single scalar value that will represent a player's quality. We cannot let the fitness of a player be represented by some weighted sum of its component dimensions, for example. This is because the value of the weighting for an aspect of play may depend on who the opponent is; for one opponent, **x** may be more important than **y**, for another opponent maybe only **y** is important. This subjectivity prevents us from reducing a multi-dimensional player to a single scalar and then determining a winner by comparing these values.

A simple way to model these aspects of coevolution is to allow some comparison between individuals to determine a single dimension that will, for these individuals, determine the outcome of the match. One way to do this is to choose that dimension in which the two players are most distinct. We define $f2(a,S)$, where a and each member of S are pairs, as follows:

$$f2(a,S) = \sum_{i=1}^{|S|} score2(a,S_i) \qquad \textbf{/eq.2}$$

where

$$score2((a_{\mathbf{x}},a_{\mathbf{y}}),(b_{\mathbf{x}},b_{\mathbf{y}})) = \begin{cases} score(a_{\mathbf{x}},b_{\mathbf{x}}), \text{ if } (|a_{\mathbf{x}}-b_{\mathbf{x}}|>|a_{\mathbf{y}}-b_{\mathbf{y}}|) \\ score(a_{\mathbf{y}},b_{\mathbf{y}}), \text{ otherwise.} \end{cases}$$

and, as before, $score(a,b)$= 1 if $a>b$, 0 otherwise.

[1] Notice that neither of these correspond to the Darwinian meaning of fitness relating to the number of viable offspring. Even in regular evolution the number of offspring an individual produces is regulated by the objective fitness of other individuals in the population as well as its own objective fitness.

This game is easily extended to more than two dimensions by asserting that whichever dimension has the biggest difference between opponents is the dimension that determines the outcome of the game. Note that the game has the desirable property that a generalist, a player that is maximal in all dimensions, can be defined that beats all other players. Accordingly, we assert that the objective of this game is to maximize all dimensions – i.e. the objective fitness of an individual is the sum of all dimensions. Potentially, a coevolutionary set-up could enable selective pressure to move from one dimension to the other dimension focusing on whichever is weakest. On the other hand, it might focus on one dimension to the detriment of other dimensions. We will use this game to model the effects of focusing and over-specializing that can occur in coevolution.

2.4 INTRANSITIVE SUPERIORITY

In Section 2.2, we considered the case where coevolution is erroneous in determining the superiority of two individuals when each is compared to some other sample of individuals. However, when using coevolutionary games, it is possible to create problematic scenarios even when comparing individuals against each other.

For example, it is quite conceivable that for three chess players, *A*, *B* and *C*, *A* can reliably beat *B*, *B* can reliably beat *C*, but *A* cannot beat *C*. Simply stated, we may say that the superiority of players in chess is not transitive. Further, suppose that *A* may be beaten by *C* creating a loop as in the "rock, scissors, paper" game – we might call this a game with *circular superiority relations* or *circular dominance relations*. This may result in local superiority relationships that provide a deceptive gradient and encourage strategies that are inferior in a global sense (e.g. further away from some strategy *D* which beats *A*, *B* and *C*). Or, coupled with over-specialization, coevolving species may drive each other from strategy to strategy, apparently improving, only to arrive back where they started.

The concept of intransitive superiority is central to issues in coevolutionary failure (Cliff & Miller 1995), and we want to be able to include it in our minimal substrate. To do this we will have to use at least a two dimensional game. Consider: if all the relevant characteristics of a player can be represented by a single value - for example, the ability of a javelin thrower can be characterized by distance alone - then such circular dominance is not possible. But in fencing, for example, the ability of a player is multi-dimensional including for example, the ability to parry, the ability to thrust, and stamina. As already stated, we cannot simply sum the ability of the swordsman in each of these respects - which of these characteristics is critical, or the weighting of these characteristics, depends on the characteristics of their opponent. In such cases where the ability of a player is multi-dimensional it is quite possible that three or more players may form a circular superiority relation.

A simple way to modify our game to incorporate intransitive superiority is to modify Equation 2 so that the dimension that determines the outcome of a game is the dimension in which the players are most similar (instead of most different). That is, when two players, $(a_{\mathbf{x}},a_{\mathbf{y}})$ and $(b_{\mathbf{x}},b_{\mathbf{y}})$, enter a game the winner will be whoever is the greater in the dimension in which they are closest.

$$f3(a,S)=\sum_{i=1}^{|S|} score3(a,S_i) \qquad \textbf{/eq.3}$$

where

$$score3((a_{\mathbf{x}},a_{\mathbf{y}}),(b_{\mathbf{x}},b_{\mathbf{y}})) = \begin{cases} score(a_{\mathbf{x}},b_{\mathbf{x}}), \text{ if } (|a_{\mathbf{x}}-b_{\mathbf{x}}|<|a_{\mathbf{y}}-b_{\mathbf{y}}|) \\ score(a_{\mathbf{y}},b_{\mathbf{y}}), \text{ otherwise.} \end{cases}$$

and, as before, $score(a,b)= 1$ if $a>b$, 0 otherwise.

Using this game we can easily define three players that exhibit circular superiority, *a* beats *b* beats *c* beats *a*. For example, $a=(1,6)$, $b=(4,5)$, $c=(2,4)$: *a* beats *b* because they are closest in the **y** dimension and $a_y>b_y$; *b* beats *c* because they are closest in the **y** dimension and $b_y>c_y$, but *c* beats *a* because they are closest in the **x** dimension and $c_x>a_x$.

Note that this game still has the desirable property that a player that is maximal in both dimensions beats all other players. Again, we assert that the objective fitness of an individual in this coevolutionary game is the sum of all dimensions.

3 EXPERIMENTAL SET-UP

The following experiments use the games defined in Equations 1 through 3. In addition to defining the game we will use there are several other choices to be made in the set-up of the coevolution:

- Number of populations (who competes with who?, who reproduces with who?)
- Choosing members to make *S* (who plays who?)
- Sample size (how many do you play?)
- Selection scheme
- Variation operators

The following experiments will use one or two separate populations. Selection and reproduction in one population will operate independently of the other population in the cases where there are two. Unless otherwise stated, the population size is 25 for each population. In principle, the choice of who plays who is independent from the segregation of reproduction. However, in the following experiments when there is more than one population we shall limit ourselves to considering the case where players only play against opponents from the other population. Unless otherwise stated the sample size, $S=15$. We use fitness proportionate selection, and for simplicity we use mutation as the only variational operator. One detail we found illuminating concerns the bias of the mutation operator.

3.1 MUTATION BIASES

Because we are using such a simple substrate we must be careful about the assumptions we make with respect to the likelihood of beneficial and detrimental variations. If we imagine that our individuals are represented by real numbers then we might reasonably assume that a mutation would be equally likely to increase or decrease the value - perhaps we would add a random value drawn from a Gaussian distribution. If, alternatively, we were to represent individuals using a unary representation (simply the unitation, number of ones, of a fixed-length binary string) and vary values by mutating bits then mutation would have inherent biases. Specifically, a string with more than half zeros is more likely to increase than decrease, a string with more than half ones is more likely to decrease than increase, and in general, there is a natural bias towards strings with 50:50 ones and zeros.

In real applications, for example, a neural network controller, sorting networks, or a genetic programming game player, there are likely to be significant mutational biases. It may well be the case that a random neural network controller, sorting network or genetic program is likely to be superior to a null or default representation that might be used to initialize individuals; for example, a neural network with no connections or weights of 0, a sorting network with no comparitors, or a GP tree with no nodes. However, once a moderate solution has been found we would reasonably expect the situation to change. In the later stages of evolution it is likely to be the case that nearly all changes to an individual will be detrimental. We will call this situation a *negative mutation bias*. These basic observations have theoretic underpinnings in the simple models used by Fisher (1930).

Since we are abstracting an evolutionary substrate to a scalar (or two) we must be careful with assumptions like unbiased mutation. The following experiments will use a biased mutation. A simple way to do this is to evolve integers as if they were represented with a fixed length binary string and the value they represent is given by the unitation of the string. The (simulated) string length will be 100 and mutation per bit will be 0.05 probability of assigning a new random value.

4 EXPERIMENTS AND RESULTS

We start with a control experiment, and then conduct several experiments using Equations 1 through 3 to illustrate a few of the concepts we have discussed.

4.1 CONTROL: MUTATION BIAS

The first experiment is a control experiment using $f(a)=0$ to illustrate the effects of mutation bias and provide reference performance levels for the following experiments. We evolve single integer values in two separate populations with the biased mutation discussed previously. All individuals in the first population are initialized to 0. All individuals in the second population are initialized to 100. Figure 1 shows the populations evolving over time. The vertical axis represents the objective fitness of individuals. Reference lines are included at 50 and 100. The horizontal axis runs from generation 0 to generation 600.

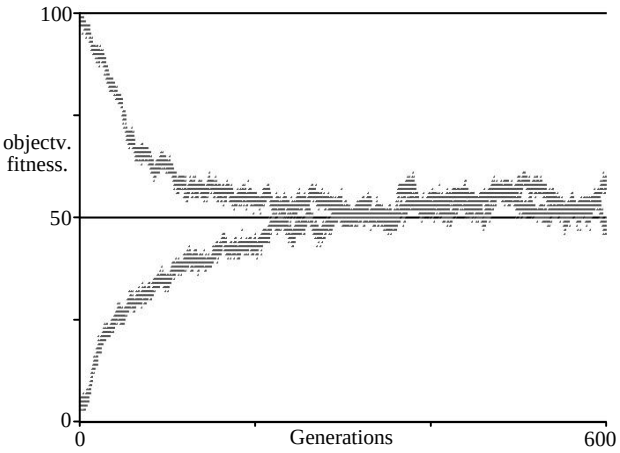


Figure 1: Neutral selection showing mutation bias.

We see that the population average is drawn to about 50 in both cases, as predicted. The performance level in the latter half of the run represents a neutral, no-selection, performance level for the populations.

4.2 EXPERIMENT 1: LOSS OF GRADIENT

Next we demonstrate that the subjective measure of fitness does not always deliver the desired objective performance even in the simple one-dimensional game of Equation 1. Figure 2 shows the performance of individuals in two coevolving populations.

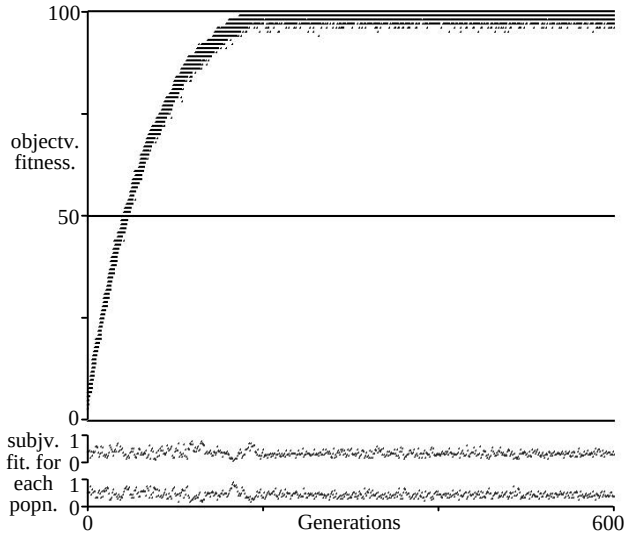


Figure 2: Coevolution using Equation 1.

The additional two plots in the lower section of the figure show the average subjective fitness for members of each population. These show that the subjective fitness of one

population is approximately one minus the subjective fitness of the other. They also show that the subjective fitness of either population does not vary significantly as evolution progresses despite the fact that objective fitness has changed. This is the Red Queen effect (Cliff & Miller 1995) – though the performance of individuals improves, the performance of their opponents improves at the same rate, and they find themselves no better off (subjectively). In an experiment where we do not have an objective measure of performance, this creates a problem for monitoring progress.

Figure 3 shows the same experiment but with a sample size, S=1. i.e. each player is evaluated by playing against one randomly selected player from the other population.

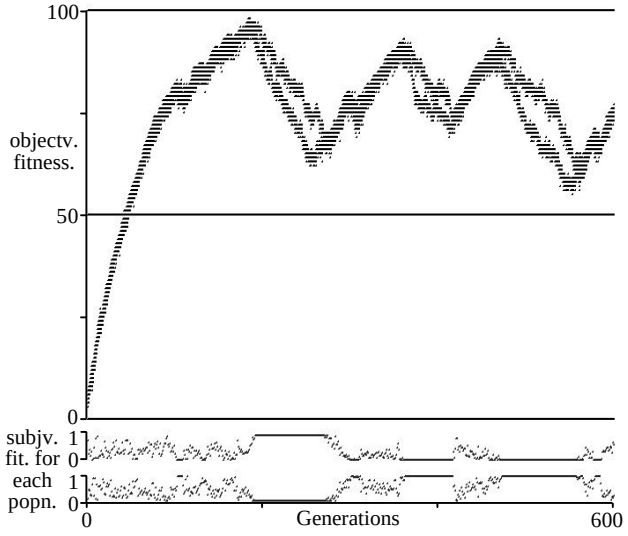


Figure 3: Coevolution using Equation 1, S=1.

This is clearly a different result altogether. There are clear downward trends as well as upward trends. Notice that the subjective fitnesses (at the bottom of the figure) show periods of polarization – one population scores 1 and the other population scores 0 – and these periods coincide with the downward trends in objective fitness. At these times all the individuals in the first population beat all the individuals they are tested against in the second population, or vice versa. This separation of the populations can be seen in the points plotted for the objective fitness values. Thus there is no selective pressure and the negative mutation bias is thus allowed to pull the population back down towards the neutral performance position shown in Figure 1. Then, by chance, the two populations happen to re-engage and race each other to high values again. This may happen repeatedly in a run.

In this game, the effect is only seen at these low population sizes and low sample sizes, and the good performance seen in Figure 2 can be regained using a larger population size, even with S=1. However, it is surprising that such a disconnection of the populations is possible at all in such a simple symmetric game. In a practical application of coevolution the likelihood of one population dominating the other may be affected by asymmetry in the game – for example, evasion may be easier than pursuit, and a population of evaders may get a little too far ahead on occasions, and cease to provide selective pressure. But note that even if the coevolving populations do not disconnect completely as they do here, the subjective fitnesses may be distorted.

4.3 EXPERIMENT 2: FOCUSSING

In these experiments we use Equation 2 to illustrate problems of focusing. We have already seen two populations coevolving successfully on a single dimension in Figure 2. Figure 4 shows two populations evolving on ten dimensions. To avoid using a larger genome, that would suffer unfairly from our mutation bias, we use ten dimensions of 10 bits each (instead of one dimension of 100 bits). The vertical axis shows the objective fitness of each individual, i.e. the sum over all dimensions.

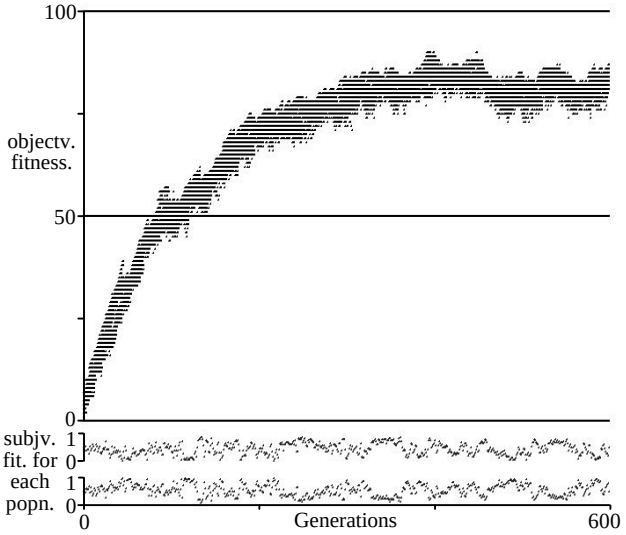


Figure 4: Coevolution using Equation 2, 10 dimensions.

Notice that the performance levels fail to reach 100. This can be explained by noticing that whilst any one dimension is the dimension that matters given the make-up of the other population, the other nine dimensions are likely to drift toward their neutral position. Although selective pressure switches from one dimension to another, high performance cannot be maintained in all dimensions simultaneously. This may cause individuals to 'forget' skills that they had learned previously, only to re-discover them later.[2] If our objective metric was concerned with only a subset of the ten dimensions then oscillations in performance would be pronounced. But, even when the objective metric values all dimensions equally we see that over-specializing can prevent the discovery of a generalist. Depressed performance also occurs in single-population coevolution using this game (drawing S from other members of the population).

[2] This is clear in these experiments when the performance in each dimension is observed separately (not shown).

In a normal evolutionary set-up, the failure to reach the maximum performance could be remedied with the use of elitism in the objective metric. But, note that elitism acting in the subjective metric cannot assist us here. We only have access to the 'apparent best' and elitism in this metric will not produce elitism in the actual (objective) best. However, a "Hall of Fame" method, where individuals play against representatives from past generations may decrease 'forgetting' and increase generalization (Cliff & Miller 1995).

4.4 EXPERIMENT 3: RELATIVISM

In our third experiment we examine the game in Equation 3 that exhibits intransitive superiority. Figure 5 shows the intransitive game with two populations.

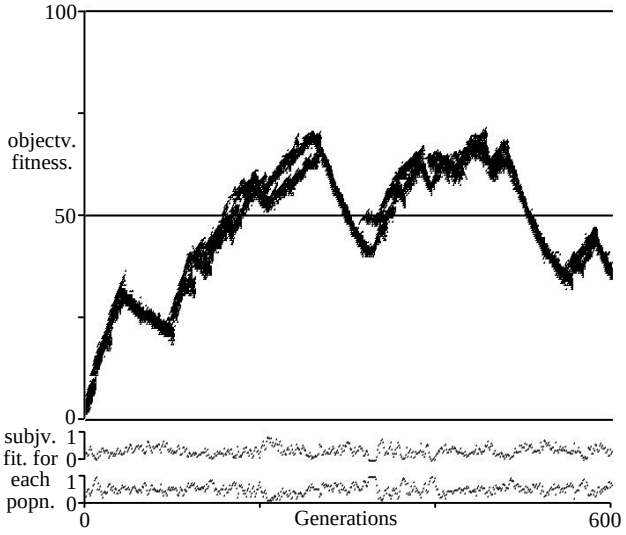


Figure 5: Coevolution using Equation 3.

Clearly, things are not working the way we want them to according to our objective metric. Notice that the downward trends are not accompanied by a domination of one population by the other – the average subjective fitnesses are not polarized as they are in Figure 3. So the downward trends are not the result of drifting under negative mutation bias. Also, the downward excursions sometimes go below the neutral level of 50 showing that the populations are actually being driven downwards.

This activity can be explained by noticing that subjective scores in this game can sometimes be improved by lowering the value a player represents. Specifically, if a player is losing in the chosen dimension it may be possible to change which dimension is relevant by lowering its value. In some circumstances, this may make the second dimension become relevant and the outcome of the game may be different. For example, consider a=(4,7) and b=(5,5). The closest dimension is the first, and b wins. Now, a'=(3,6) is a small variation from a. The closest dimension when a' plays b is the second dimension and a' wins. So, a' is preferred over a even though a' is inferior to a in the objective metric.

So, whereas Figure 3 showed how subjective preference may give a draw where objective preference should give a winner, in this experiment, we see that P_{subj} may give the opposite answer to P_{obj}. As a result, we see that performance can be *driven* down not just *drift* down. This dynamic is produced by the exact characteristics of the game we defined. However, it is sufficient to illustrate the point that P_{subj} can be the reverse of P_{obj} even in a game which looks innocent enough. The difficulty that Equation 3 causes arises from the fact that the features of a player that control a win with respect to one player, are in opposition to the features that will win against another player. Specifically, reduction in some dimension can allow a win against one player, whilst inducing a loss against another player. We may expect such destructive dynamics in any game with these counter acting properties.

Figure 6 shows that the effects of intransitive superiority can be destructive even in one-population coevolution. That is, even when players play against opponents from their own population, the intransitive nature of the game can prevent continued increases in performance. Interestingly, the data from this run can be seen to exhibit some 'spontaneous speciation'. Although there is only one population, the individuals occasionally diverge showing two separate sub-populations.

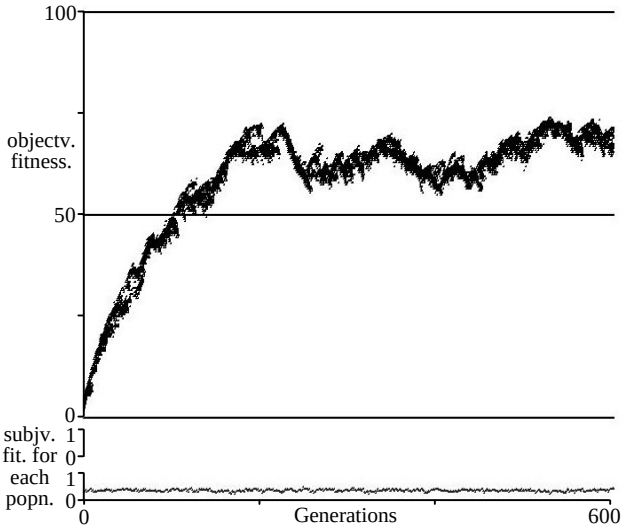


Figure 6: Coevolution using Equation 3: one population.

The phenomena in these experiments with Equation 3 are not overcome by larger population sizes and larger sample sizes. Examining the exact values in both dimensions (not shown in these figures) reveals that evolution in this game is indeed moving through the same parts of strategy space repeatedly. Thus this simple game illustrates the cyclic activity often speculated about in coevolution literature (Cliff & Miller 1995).

5 CONCLUSIONS

With the use of our minimal substrate we have provided concrete illustrations of several coevolutionary issues. We have given simple examples in which subjective fitness measures appear unproblematic but can actually disagree with objective fitness.

We illustrated three kinds of coevolutionary failure: *loss of gradient* where performance drifts (downward) because one population dominates the other, *over-specialization* where coevolution fails to find general solutions because strategies transition from one dimension to the other exploiting specific weaknesses, and issues of *relativism* where subjective fitness can act in opposition to objective fitness. Each of these weaknesses in coevolution can cause repeated 'forgetting' and re-discovery of strategies and prevent the continued improvement in performance that we would like to see.

Important concepts in these illustrations include the separation of *objective* and *subjective* fitness: the metric that we as researchers seek to optimize, and the metric of performance as perceived by the co-evolving individual, respectively. Also, the fact that a coevolutionary game may not be reducible to a single dimension – the performance of an individual is always *with respect to* some other individual (or set of individuals) – thus the subjective metric may not be reduced to a one-dimensional notion of quality, or a single superiority ordering. Finally, intransitivity is an important characteristic of subjective superiority that can be particularly problematic.

In illustrating these problems and concepts we have made many choices both in the game and the coevolutionary set-up. Our substrate is by no means the only simple substrate in which these concepts could be illustrated. Nonetheless, the coevolution of scalars and vectors provides one concrete example for several of the slippery issues common in the coevolution literature. And, unlike previous work, in this substrate we are able to properly separate the issues of coevolution from the issues of any complex application domain. The problems caused by these simple games caution us in making assumptions about more complex coevolutionary endeavors.

In previous work we have used the term 'mediocre stable state' to mean what we may now describe as a condition where the coevolutionary system is not producing improved performance in the objective metric despite continued adaptive steps in the subjective metric. This paper has begun to decompose the mechanisms that may be behind such failures, and in so doing, it may assist us in at least diagnosing problems in future. Related work builds upon the insights here to formulate an optimization method that explicitly respects the multi-dimensional nature of coevolutionary games by applying the notions of multi-objective optimization to a set of subjective scores.

Acknowledgments

The first author would like to thank the (past and present) members of DEMO at Brandeis, notably Alan Blair, Hughes Juilles, Paul Darwen, and Greg Hornby, but especially, Sevan Ficici for numerous invaluable discussions about the nature of coevolutionary difficulty.

References

Cliff, D, & Miller, GF, 1995, "Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations", *Third European conference on Artificial Life*, pp 200-218, Springer-Verlag, LNCS 929.

Ficici, SG. & Pollack, JB., 1998, "Challenges in Coevolutionary Learning: Arms-Race Dynamics, Open-Endedness, and Mediocre Stable States." *Proceedings of the Sixth International Conference on Artificial Life*. Adami, et al, eds. Cambridge: MIT, Press.

Fisher, RA, 1930, *The genetical theory of natural selection*, Clarendon press, oxford.

Goldberg, DE, 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading Massachusetts, Addison-Wesley.

Hillis, DW, 1992, "Coevolving parasites improve simulated evolution as an optimization procedure", In Langton, C, ed., *Artificial Life II*, Addison Wesley.

Holland, JH, 1975 *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: The Uni. of Michigan Press.

Juille, H, & Pollack, JB, 1996, "Coevolving intertwined spirals", *procs. of fifth annual conference on evolutionary programming*, pp461-468. MIT press.

Juillé, H. & Pollack, JB, 1998, "Coevolving the 'Ideal' Trainer: Application to the Discovery of Cellular Automata Rules." *Proceedings of the Third Annual Genetic Programming Conference*, Madison, WI, July.

Kauffman, S, 1993, *The Origins of Order*, Oxford University Press.

Maynard-Smith, J., 1982, *Evolution and the Theory of Games.* Cambridge University Press, Cambridge.

Miller GF, & Cliff D, 1994, "Protean Behavior in Dynamic Games: Arguments for the Co-Evolution of Pursuit-Evasion Tactics" in Cliff, D, Husbands, P, Meyer, JA, and Wilson, WS, eds. *From Animals to Animats 3: Procs. of Third International Conference on Simulation of Adaptive Behavior* (*SAB94)*. MIT Press, pp.411-420.

Noble, J, & Watson, RA, "Pareto Coevolution: Using performance against coevolved individuals as dimensions for pareto selection", Spector, L, et al, editors. GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, 2001.

Pollack, JB, Blair, A. & Land, M. 1996, Coevolution of A Backgammon Player. *Proceedings Artificial Life V*, C. Langton, (Ed), MIT Press.

Reynolds, CW, 1994, Competition, Coevolution and the Game of Tag, in the *proceedings of Artificial Life IV*, R. Brooks and P. Maes, Editors, MIT Press, Cambridge, Massachusetts, pp 59-69.

Sims, K, 1994, "Evolving 3d morphology and behavior by competition", in Brooks & Maes, eds., *Artificial Life IV*, pp28-39, MIT press.

Probing the Persistent Question Marks

Janet Wiles
School of CS&EE and
School of Psychology
The Univ of Queensland
Brisbane, QLD 4072
janetw@csee.uq.edu.au

Ruth Schulz
School of CS&EE
The Univ of Queensland
Brisbane QLD 4072
ruth@csee.uq.edu.au

Jennifer Hallinan
Institute for Molecular Biosciences
The Univ of Queensland
Brisbane QLD 4072
j.hallinan@imb.uq.edu.au

Scott Bolland
Bradley Tonkes
School of CS&EE
The Univ of Queensland
Brisbane QLD 4072
scottb@csee.uq.edu.au
btonkes@csee.uq.edu.au

Abstract

The puzzle of the persistent question marks (a phrase coined by Harvey in 1993) refers to the occurrence of residual learnable alleles in simulations of the Baldwin effect, which concerns interactions between learning and evolution. Explanations of the puzzle have focused on the role of fitness proportional selection pressures and of drift. We replicated the original 1987 model by Hinton and Nolan, extending the number of generations simulated until populations converged on stable genotypes, thus quantifying the number of residual question marks and the time to homogeneity (only one allele remaining in the population for each gene in a chromosome). Previous explanations of the residual question marks imply that algorithms that maintain strong selection pressure should not result in residual question marks. We tested this implication by simulating Hinton and Nolan's Baldwin model replacing frequency proportional selection by tournament selection. Fewer residual question marks remained, but they were still present in many populations. We analyzed the relative factors that contribute to these effects in both fitness proportional and tournament selection runs. We conclude that homogeneous question marks are a significant factor in populations exhibiting the Baldwin effect in all the types of selection strategies tested in this study.

1 INTRODUCTION

Learning is ubiquitous in nature. It may be argued that the ability to learn is the single most valuable trait in *Homo sapiens*, but even single-celled microbes learn to follow chemical gradients. The nature of the interaction between evolution and learning has been a subject of research and debate for decades. The suggestion that learning by individuals may guide the evolution of a population was proposed by Baldwin towards the end of the nineteenth century (Baldwin, 1896), and has been dubbed the "Baldwin Effect".

1.1 THE BALDWIN EFFECT

The assumption underlying the Baldwin effect is that behaviour may be either innate (genetically determined) or learned. Innate behaviour has the advantages of being available to the individual throughout its life, and being quick to carry out, requiring no decision-making on the part of the individual. Innate behaviour is, however, inflexible; if a novel situation arises it may not be dealt with appropriately. Learned behaviour is exactly the opposite: slower to acquire and apply, but flexible.

Baldwin suggested that some individuals may contain genes which predispose them to learn a particular behaviour. If this behaviour is advantageous to the individuals, they will produce a higher than average number of offspring, and the advantageous genes will spread through the population. Further mutations to these genes are then possible, making the behavior more innate and hence easier to learn. Eventually a behavior that was learned may become encoded into the genome as an instinct. Learning on the part of individual organisms has then shaped the evolution of the entire population.

1.2 MODELLING THE BALDWIN EFFECT

Baldwin's hypothesis, like those of Darwin, Lamarck and other evolutionary theorists was not testable with the science of the nineteenth century. It remained an intriguing possibility until the advent of sufficiently powerful computers and evolutionary computation algorithms made empirical investigations into evolutionary processes a real possibility, almost a century later.

The most widely studied model of the Baldwin Effect was developed by Hinton and Nowlan (1987). In this model, each individual consists of a simple neural network with twenty connections, which must be set correctly via either

learning or evolution. A network that achieves the correct settings has a fitness dependant upon how long it took to achieve the correct settings, while all incorrect networks have equal, minimal fitness. The genetic search space is thus a "needle in a haystack", with the ability to learn smoothing the single spike of high fitness (Figure 1).

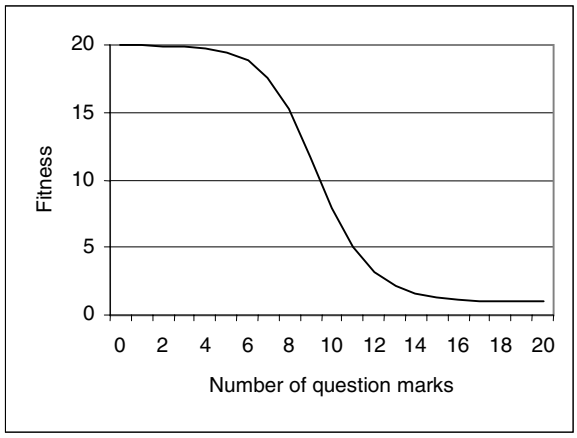


Figure 1. Search space for the Baldwin Effect, viewed in terms of expected fitness. The highest fitness is given for a genome consisting of twenty 1s, which only occurs in one in a million randomly generated individuals (a needle in a haystack problem). By allowing some genes to be learned (the ?s), the landscape around the single target of high fitness develops "shoulders'' that allow effective searching.

Computationally, each individual is implemented as a string of twenty "genes", each of which may be either 1, 0, or ? (question mark). The ? represents a learnable gene. The individual learns by guessing 0 or 1 with a probability of 0.5. The target pattern is a string of twenty 1s. The number of guesses required to achieve this target, g, is recorded and used to calculate the individual's fitness, f:

$$f = 1 + \frac{(L-1)(G-g)}{G}$$

where G is the maximum number of guesses allowed and L is the length of the chromosome. In Hinton and Nowlan's model, G = 1000, L = 20, and the population size, N = 1000.

Hinton and Nowlan (1987) modeled the Baldwin Effect using a simple genetic algorithm, with no mutation and a crossover value of 1.0; on average each pair of parents undergoes crossover once during each reproduction event. They state that the probability of an individual being selected as a parent for the next generation is proportional to the fitness of that individual, with an organism that learns immediately being twenty times as likely to be selected as an organism that never learns. Such a selection strategy is usually implemented using the roulette wheel algorithm (Mitchell, 1996). The next generation is created by repeatedly selecting two parents for each new individual. The probability that an individual is selected as a parent is proportional to its fitness divided by the total population fitness.

Hinton and Nowlan (1987) demonstrated that under these conditions "learned" behaviours, represented by 1s, did in fact become genetically encoded, rising from an initial 25% of alleles in the population to over 50%. Non-target alleles, represented by 0s, disappeared from the population and the proportion of learnable alleles, represented by question marks, reduced slightly from an initial 50% of the alleles in the population. A typical run demonstrating the Baldwin Effect as modeled by Hinton and Nowlan is shown in Figure 2.

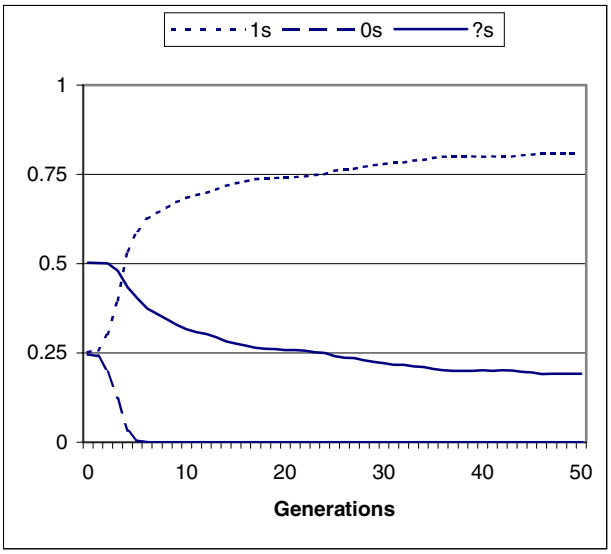


Figure 2. Replication of the Baldwin Effect simulations by Hinton and Nowlan (1987). The alleles in the initial population are generated with probability 50% learnable (question marks), 25% target (1s) and 25% non-target (0s). The 0s are rapidly removed from the population, the 1s increase and the ?s decrease. In this simulation, the proportions of alleles appeared to stabilize at approximately 80% 1s and 20% question marks (but see text for further discussion of the convergence properties).

1.3 THE PERSISTENT QUESTION MARKS

An interesting feature of the Hinton and Nowlan model is the persistence of learnable genes, represented by question marks, in the population once it has stabilized. Hinton and Nowlan themselves suggest that this is because there is very little selective pressure in favor of

genetically specifying the last few connections, since learning can occur in very few guesses.

Belew (1987) carried out a detailed analysis of the Hinton and Nowlan model. He concluded that there are three causes for the persistence of question marks: little selection pressure for replacing the last few question marks; the fact that a discrete number of offspring means that a slight selective advantage translates merely into a slightly higher probability of producing an extra offspring; and the fact that the number of offspring produced per parent depends on the fitness differential between the individual and the average population fitness, a value that will generally be extremely small.

Harvey (1993) argued that Belew's interpretation is erroneous; he suggests that "the combination of genetic drift and hitch-hiking so completely swamps the selective pressures that some of the genes are completely converged to the undecided value, rather than the 'correct' one" (Harvey, 1993, p. 1).

Hinton and Nowlan, Belew and Harvey agree fundamentally, that the explanation for the persistent question marks lies in the reduced importance of selection pressure compared with other factors at work in the GA late in the course of evolution. All of the mechanisms described by these researchers are affected to a large extent by the choice of selection operator in the model. Following this reasoning, a selection operator with different characteristics, such as the ability to exert a strong selection pressure when small differences in fitnesses exist, should result in different equilibrium behaviour in the model. In this paper we re-examine the extent to which the persistence of question marks is due to practical, as opposed to theoretical, factors of the model.

2 METHODS

2.1 THE MODEL

The model used in these experiments was a re-implementation of the Hinton and Nowlan (1987) model, coded in Java. The initial parameters were similar to their values with initial proportions of 1, 0 and ? alleles 0.25, 0.25 and 0.50 respectively. We made a minor change of setting both population size and maximum number of guesses to 1024 instead of 1000, for comparison with other population sizes that were powers of two.

The two major factors to be investigated in determining the persistance of question marks in the equilibrium population of this model are

1. Genetic drift; and
2. Selection pressure.

In order to dissect the effects of these factors, we ran multiple runs until convergence was reached. Hinton and Nowlan's original simulations were run for 50 generations, and Harvey's for 500 generations. In many replication runs, at these points the populations are still changing, albeit slowly. Given sufficient time, runs converge to a stable population, in which all individuals in a population have identical genotypes. The fitness values may still fluctuate slightly as residual question marks result in slightly varying fitnesses for each generation, but the composition of the genotype in a population stabilizes when all genes are homogeneous (only one allele remains in the population for each locus on the chromosome). In our simulations, the number of generations was chosen so that a substantial proportion of runs had converged to homogeneous populations.

In order to study the long term behaviour of the model using roulette wheel selection and the variance in performance, we simulated 100 runs using the parameters described above. The number of runs which became homogeneous for all genes, the generation at which homogeneity occurred, and the number of persistant question marks in each population were recorded (section 3.1).

To assess the contributions of genetic drift and selection pressure, comparison simulations were run with no selection pressure (sections 2.1.1 and 3.2), and an alternative selection algorithm (sections 2.1.2 and 4).

2.1.1 Genetic Drift

Genetic drift is defined by Maynard Smith (1998) as fluctuation in the proportions of different kinds of individuals due to chance. Drift may result in the elimination of some alleles from a population, and in others becoming homogeneous, depending on the population size.

In a genetic algorithm, drift would be expected to play a significant role only in the outcome of simulations with a small population size. Harvey (1993) is of the opinion that the population size used by Hinton and Nowlan (1987), 1000 individuals, is small enough for drift to be important. In practical EC studies, population sizes much less than 1000 are often used, so the role of drift in populations of this size are of importance not just for the Baldwin effect, but potentially for other studies also.

In the Baldwin model, a run that results in one or more homogeneous 0s in the population would have to reflect the result of chance, since no selection pressures favor 0s. By contrast, both homogeneous 1s and question marks are subject to selection pressures that favor their increase. For the 1s, selection is always favorable, and ?s are favored over 0s at the start of runs, before the 0s have been eliminated from the population.

In order to study the role that drift plays in the convergence to homogeneity, we simulated 100 runs of the model using the parameters detailed above with the sole change that no selection pressure was applied (i.e., selection of parents was random). The same measures were recorded as for the first set of simulations (section 3.2).

2.1.2 Selection Pressure

Previous investigators of the Baldwin Effect agree that fitness-proportional (roulette wheel) selection leads to reduced selection pressure against question marks as the population stabilizes. This effect arises because roulette wheel selection used the fitness differential between individuals to compute the expected number of offspring. As the total number of question marks in the population is reduced, the fitness differential between individuals is exponentially reduced. For example, the maximum fitness for an individual in the Baldwin model is 20. For one learnable allele the expected fitness is 19.96 and for two learnable alleles it is 19.92. In a population in which half the agents had two learnable alleles and the other half had one learnable allele, the expected proportions of the two groups as parents for the next generation would be 50.05% with one learnable allele, and 49.95% with two.

An alternative selection strategy is tournament selection. In this algorithm two individuals are selected at random from the parent population, and the individual with the higher fitness becomes a parent. The probability of being selected as a parent for the next generation is therefore dependant upon the relative rank of an individual within the population, rather than its proportional fitness. Under tournament selection, the reduced fitness differential later in evolution does not change the ranking of individuals and selection pressure is maintained as long as there are different fitnesses within the population. Consider the example given above. The expected *fitness* of individuals with zero, one or two learnable alleles is the same as in roulette wheel (20, 19.96 and 19.92 respectively), however, the selection of parents is by competition, so even these very small differences confer advantage. Consider the differences this change in the algorithm makes for the example given above. In a population in which half the agents had two learnable alleles and the other half had one learnable allele, with tournament selection, the expected proportions of the two groups as parents for the next generation would be 75.00% with one, and 25.00% with two learnable alleles.

Thus, under tournament selection, we expect to find strong pressure to remove all residual question marks from the population. Any divergence from complete homogeneity of 1s would indicate the presence of other factors at work.

We ran 100 runs of the model using tournament selection, but with all other parameters as above for roulette wheel selection. The same sets of measures were recorded (Section 3.3).

3 RESULTS AND DISCUSSION

3.1 PROPORTIONAL FITNESS SELECTION

The replications of the roulette wheel model show that at 50 generations (the number of generations used by Hinton and Nolan, 1987), all runs had residual question marks, which persisted for many subsequent generations. By 500 generations (the number of generations used by Harvey, 1993), the proportions of alleles had changed, and the populations had far fewer learnable alleles, with all runs having 20% or fewer residual question marks.

However, when allowed to continue beyond 500 generations, all populations either converged on a single genotype (i.e., reaching homogeneity in all genes) or appeared close to convergence. By 2500 generations, 85/100 populations had converged to homogeneity and the remaining 15 populations had 19/20 homogeneous genes. No 0s remained in any of the populations, and the number of residual question marks varied from 0 to 4 per individual (out of the total of twenty genes). The average number of residual question marks in the converged runs was 1.63 (standard deviation 1.02).

By studying the distribution of residual question marks in the homogeneous populations, the issue of the persistent question marks may be examined in detail.

Runs that took less than about 500 generations had more residual question marks than those that took longer to converge. The distribution can be seen by graphing the time taken to reach homogeneity against the number of residual question marks (see Figure 3).

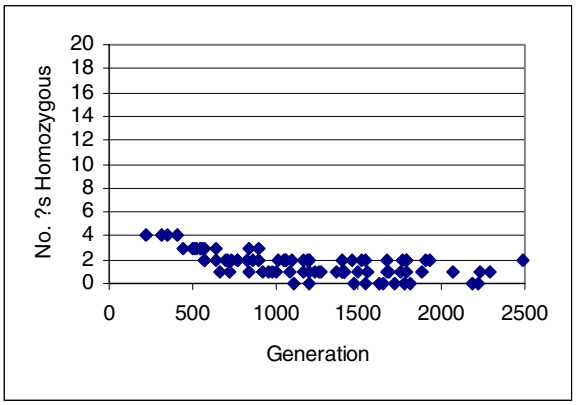
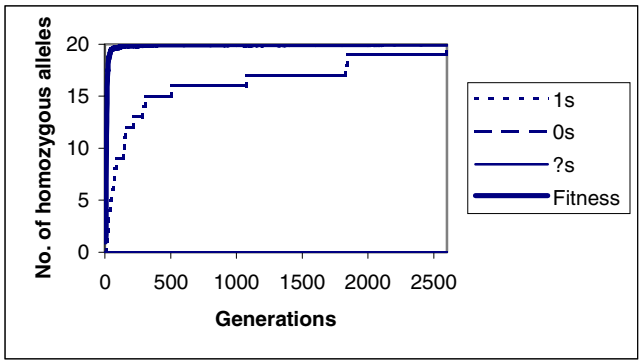
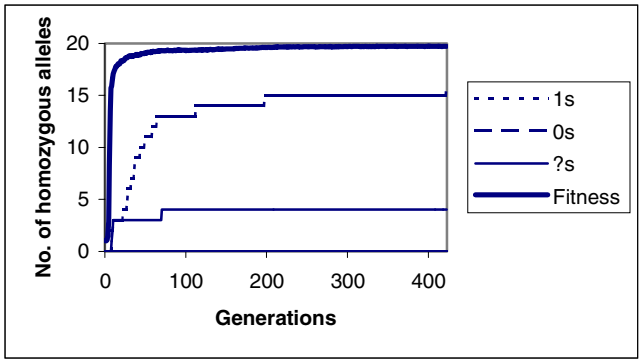


Figure 3. Time to converge for roulette wheel runs. When runs converged quickly (less than 500 generations), they tended to have more residual question marks.

Inspection of individual runs revealed that the average fitness of the population started at the minimum value (1.0) as expected, and at about twenty generations rose steeply to the high teens. It then gradually improved over the following hundreds (in some cases thousands) of generations until the population reached homogeneity (See Figure 4). This behavior - a steep initial rise in fitness followed by slow convergence - is consistent with the thesis of diminishing selection pressures.



(a)



(b)

Figure 4. Fitness and number of homogeneous genes vs generation for runs with zero (a) and four (b) homogeneous question marks. Both runs show a steep early rise in fitness and then variable lengthy periods before convergence (note the difference in scales on the x-axis). The steep rise in average fitness occurs before any genes become homogeneous in (a), and all genes are eventually 1s. In (b), by generation seven, fitness is more than 50% of the maximum and by generation ten three loci have homogeneous question marks.

The long convergence time (after the initial rapid rise in fitness) suggests that drift may be the only mechanism affecting the incidence of residual question marks. If drift was the sole factor in the resulting homogeneity of 1s and question marks in each population, then the distribution of question marks should have no observable pattern. By studying the inheritance of alleles throughout the generations, however, we observed that in the vast majority of cases, the residual question marks in the final population were in the same loci as those of the first successful individual in each run. This effect can be understood by considering the mechanism of inheritance in proportional selection.

In the initial population all alleles are likely to be represented in all positions on the chromosomes. The first individual to successfully guess the solution during its lifetime, on average, will have a much larger number of descendents in later generations compared to other individuals in its generation. Future winners are likely to be drawn from among these offspring, and in time, the alleles of the first individual dominate the distant descendents. This effect is similar to the "founder effect", observed when a few individuals establish an isolated population, and we refer to it as a *pseudo-founder effect.* The question mark alleles of the first successful individual are "hitch-hikers" on its success (Harvey, 1993).

3.2 THE ROLE OF GENETIC DRIFT

In the second set of simulations, parents for each generation were chosen without considering their fitness. None of the 100 runs eliminated all 0s from the population (compared to 100% elimination of 0s under roulette wheel selection). The pattern of 0s, 1s and question marks in the populations reflected the initial allele frequencies. The presence of homogeneous 0s in all runs emphasizes the effects of the strong selection pressures against them in roulette wheel runs (and has relevance to the tournament runs in the next section).

By 2500 generations (the point at which the results in the roulette wheel runs were measured above), only six of the populations had converged to homogeneity, although on average, 83.6% of the genes were homogeneous. By 7000 generations, 92% of the populations had converged (see Figure 5).

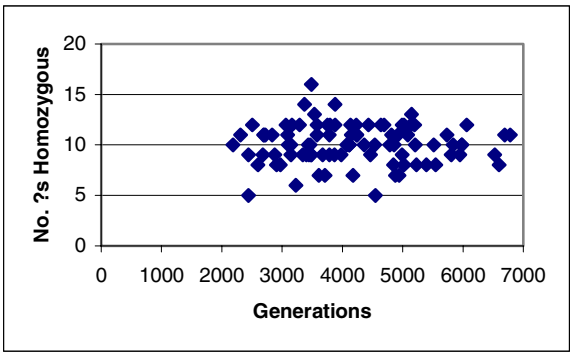


Figure 5. Time to converge for runs without selection pressues. Note the length of time to converge and the scatter of values. None of these runs eliminated 0s from the population.

We also tested drift in a variety of much smaller populations, from 32 to 512 individuals. As expected, convergence in smaller populations was much more rapid, with populations of size 32 (individuals with ten genes) converging to homogeneity in 104 generations on

average. By comparison, populations converged in 29 generations on average with roulette wheel selection.

4 SELECTION PRESSURE

The original posing of the puzzle of the persistent question marks concerned why learnable genes remain, when after many generations they might have been expected to have been replaced by instincts. With a detailed study of the factors that affect the residual question marks using roulette wheel selection, in the third set of simulations we are in a position to consider the generality of the phenomenon. That is, to what extent are the behaviours observed in the Baldwin effect due to the formulation of the problem in terms of fitness proportionate selection, and do the same factors play similar roles under other formulations?

In our third set of simulations, we examined the effects of using a tournament-based selection strategy.

Neither of the two core phenomena of the Baldwin effect (i.e., that learning can smooth the search space of a needle in a haystack problem; and that intially learned behaviour can become instinctual over time) depend on the proportional fitness selection strategy chosen by Hinton and Nolan (1987). Under other selection strategies, the core properties are expected to be exhibited.

In tournament selection, as discussed above, any small advantage between individuals in a population maintains selection pressure, so that as a population increases in fitness, the selection pressure does not decrease for the residual question marks. Thus, consideration of selection pressures leads to the hypothesis that question marks should not persist in populations subject to tournament selection.

As expected, the majority of tournament selection runs demonstrated the Baldwin effect of the elimination of 0s from the population and a gradual increase in the number of 1s over time. However, the variance was much greater than for the roulette wheel runs. Of the 100 runs, 75 found successful solutions, all within 600 generations (cf. 2500 for some of the roulette wheel runs). The unsuccessful 25 runs all had at least one homogeneous 0 by generation 2500.

Of the successful runs, the number of residual question marks varied from 0 to 8 per individual (cf. 0-4 for roulette wheel runs). The average number of residual question marks in the converged runs was 1.00 (standard deviation 1.74).

A strong trend can be seen in the time taken to reach homogeneity with respect to the number of residual question marks. Runs that took fewer generations to converge had fewer residual question marks that those that took longer to converge (see Figure 6).

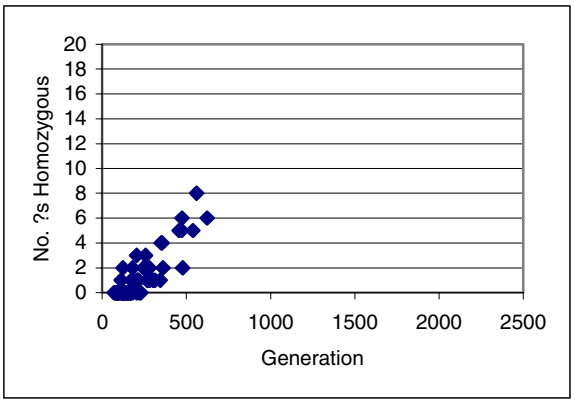
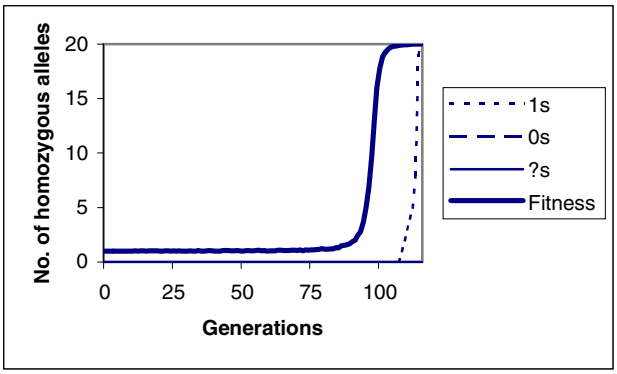
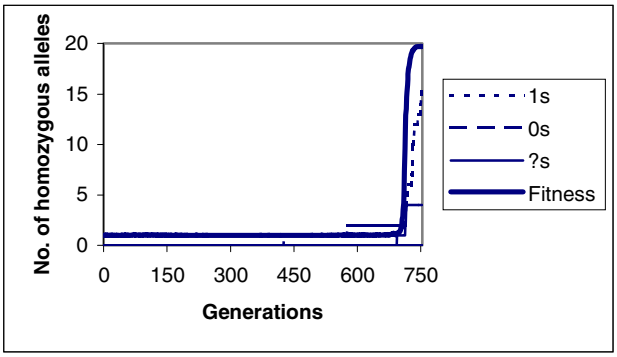


Figure 6. Time to converge for successful tournament selection runs. Compared with roulette wheel runs (Fig. 3), tournament runs show much higher variance in the number of residual question marks, time to homogeneity is much shorter and 25% of runs contained homogeneous 0s (none had converged by 2500 generations).

Inspection of individual runs revealed that the typical fitness curve over the generations had a similar shape to roulette wheel runs (initially flat, rapid rise, then flat again), but the number of generations to the rapid fitness rise varied widely (tens to hundreds of generations). After the rapid fitness rise, convergence was always fast, typically less than twenty generations (see Figure 7). The rapid convergence from high fitness illustrates the continuing effect of tournament selection pressure.



(a)



(b)

Figure 7. Tournament selection runs showing fitness and number of homogeneous genes for (a) zero and (b) four homogeneous question marks. Both runs show late rises in fitness followed by fast convergence (note the difference in scales on the x-axis). In (b), by the start of the rapid fitness rise, two loci have homogeneous 1s and one has homogeneous ?s.

The long period before the rapid fitness rise allows drift to play a role in the proportions of alleles in the populations. Some of the poorer solutions tended to have late rapid fitness rises, and frequently by that stage one or more genes had become homogeneous. In the cases where those genes were 0s, failures resulted. In the cases where several early homogeneous genes were question marks, mediocre solutions ensued.

4.1 COMPARISON OF ROULETTE WHEEL AND TOURNAMENT SELECTION

The multiple runs of the two selection strategies resulted in a majority of successful runs in both cases, but clear differences in the variance across multiple runs.

Tournament selection, which retains strong selection pressure when any question marks remain has significantly fewer residual question marks than roulette wheel (1.00 vs 1.63, $p<0.005$), supporting the basic thesis that the lack of selection pressure during the final stages of evolution under roulette wheel selection contributes to the residual question marks. However, the fact that tournament runs do have at least some residual question marks indicates that other factors are also contributing.

Interestingly, under roulette wheel selection, the runs that converged to homogeneity fastest had the most residual question marks, whereas under tournament selection the opposite effect was observed, and the runs that converged to homogeneity in fewest generations had the smallest number of residual question marks. The differences in the variance of the behaviors across different runs was marked, and can be observed directly from the graphs (cf. Figures 3 and 6).

Consideration of fitness and homogeneity over the time course of runs provides insight into the mechanisms at work in the two selection strategies.

In roulette wheel runs the average fitness of the population rises rapidly within relatively few generations (10-20) to a level at which many individuals have good, but not perfect solutions. If the population converges to homogeneity at this point, question marks from the pseudo-founders cannot be eliminated. If convergence is delayed, the large number of good solutions is able to gradually improve further, until two or fewer question marks remain.

By contrast, with tournament selection, whenever a rapid fitness improvement occurs, the population converges soon afterwards (typically 10-50 generations after the characteristic rapid fitness rise). The invariant selection pressure, independent of the reduction in the number of question marks, drives their ongoing elimination until homogeneity precludes further improvement.

Why should there be an interaction between selection strategies for the number of residual question marks and the generations required to reach homogeneity?

There are differences worth noting between the early performance of roulette and tournament selection. In tournament selection, any successful individual (regardless of the number of question marks) results in a doubling of genes in the next generation. This value is fixed, regardless of the number of question marks in its genome. Given a critical number of such genomes, the geometric increase in descendents rapidly increases the proportion of the successful individuals' alleles. However, the variance for one individual is high, and early successful individuals frequently are not sufficiently prevalent to increase in number before crossover remixes their genes with other individuals, reducing the fitness of their offspring. Before a critical mass is reached, some loci may drift to homogeneity. The variance of outcomes (higher variance in number of residual question marks and the existance of runs with residual 0s) can be traced to early homogeneous question marks and 0s before critical mass is reached.

By contrast, using proportional fitness, individuals with more 1s in their genomes have far higher rates of geometric increase (up to a factor of twenty) over individuals with far fewer. Thus, successful individuals with very high fitness can build a critical mass of descendents quickly, and those with many question marks (and hence comparably lower fitness) are less likely to become pseudo-founders. The roulette wheel runs reliably had rapid rises in fitness early in their runs (typically 20-50 generations into the run), whereas much higher variations were found with tournament selection (50-500 generations).

This difference accounts for the fact that in roulette wheel all runs succeeded, whereas in tournament selection 25% of the runs had homogeneous 0s for at least some genes.

It remains to be explained why runs with more residual question marks in tournament selection take longer to reach homogeneity. One possible explanation relates to the variance in the fitness function (as suggested by Harvey, 1993, for roulette wheel selection). Although in tournament runs the selection pressure for a given fitness value is constant, the fitness values themselves have higher variance for genotypes with more question marks, thus delaying time to convergence.

5 CONCLUSIONS

In this paper we have demonstrated the generality of the persistence of learnable alleles in the Baldwin effect using a selection strategy that does not reduce selection pressure as fitness improves. Thus, reduction in selection pressure need not be the only mechanism that results in the persistent question marks.

Roulette wheel and tournament selection were compared with respect to a variety of possible factors, including drift to early homogeneity, loss of diversity due to a pseudo-founder effect, the rate of mixing of genes and the relative selection pressures. This study demonstrates that roulette wheel and tournament selection can be analysed in terms of the time taken by a population to reach a critical mass which results in a rapid rise in fitness, and the time for a population with high fitness to converge to homogeneity. Tournament selection has fewer residual question marks on average in successful runs, but greater variance in outcomes. Roulette wheel has slightly more residual question marks, but has consistently good performance, with no populations failing to find good solutions. The study has shown that homogeneous question marks are a significant factor in populations exhibiting the Baldwin effect in all the types of selection strategies tested in this study.

In further work, we are testing the persistence of residual question marks and their underlying causes in a range of fitness proportional and tournament selection algorithms. We have been comparing selection strategies with high variance, such as traditional roulette, with low variance, such as Baker's guaranteed selection (as described in Mitchell, 1996). Results to date indicate that high variance plays a significant role in the persistance of residual question marks in populations.

Acknowledgments

This work was supported in part by a summer research scholarship from the School of CSEE to R. S., and by an Australian Postgraduate Award to S. B.

References

Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist* 30: 441 – 451. Reproduced in *Adaptive Individuals in Evolving Populations*, Belew, R. K. & Mitchell, M., eds. Proceedings Volume XXVI, Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley, Reading, MA.

Belew, R. K. (1989). When both individuals and populations search: Adding simple learning to the genetic algorithm. In *Proceedings of the third International Conference on Genetic Algorithms*, Washington, DC June 1989, pp. 34 – 41.

Harvey, I. (1993). The puzzle of the persistant question marks: A case study of genetic drift. *Computer Science Research Paper Serial No. CSRP 278*, The University of Sussex. (Also published in S. Forrest, (Ed.) Proceedings of the *Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1993.)

Hinton, G. E. & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems* 1: 495 – 502.

Maynard Smith, J., (1998). *Evolutionary Genetics*, second edition. Oxford University Press: NY.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press: Cambridge, MA.

Cyclic and Chaotic Behavior in Genetic Algorithms

Alden H. Wright
Computer Science Department
University of Montana
Missoula, MT 59812 USA
wright@cs.umt.edu

Alexandru Agapie
Laboratory of Computational Intelligence
Institute for Microtechnologies
Bucharest, PO Box 38-160, 72225 Romania
agapie@imt.pub.ro

Abstract

This paper demonstrates dynamical system models of genetic algorithms that exhibit cycling and chaotic behavior. The genetic algorithm is a binary-representation genetic algorithm with truncation selection and a density-dependent mutation. The density dependent mutation has a separate mutation rate for each bit position which is a function of the level of convergence at that bit position. Density-dependent mutation is a very plausible method to maintain diversity in the genetic algorithm. Further, the introduction of chaos can potentially be used as a source of diversity in a genetic algorithm.

The cycling/chaotic behavior is most easily seen in a 1-bit genetic algorithm, but it also occurs in genetic algorithms over longer strings, and with and without crossover.

Dynamical system models of genetic algorithms model the expected behavior or the algorithm, or the behavior in the limit as the population size goes to infinity. These models are useful because they can show behavior of a genetic algorithm that can be masked by the stochastic effects of running a genetic algorithm with a finite population. The most extensive development of dynamical systems models has been done by Michael Vose and coworkers. (See [Vose and Liepins, 1991], [Vose and Wright, 1994] and [Vose, 1999] for examples.) They have developed an elegant theory of simple genetic algorithms based on *random heuristic search*. Heuristic search theory is based on the idea of a *heuristic map* $\mathcal{G}$, which is a map from a population space to itself. The map $\mathcal{G}$ includes all of the dynamics of the simple genetic algorithm. The map defines a discrete-time dynamical system which we call the *infinite population model*.

The simple genetic algorithm heuristic $\mathcal{G}$ is called *focused* if $\mathcal{G}$ is continuously differentiable and if the sequence $p, \mathcal{G}(p), \mathcal{G}^2(p), \ldots$ converges for every p. In other words, $\mathcal{G}$ is focused if every trajectory of the dynamical system converges to a fixed point.

With one exception, infinite population models of genetic algorithms always seem to converge to a fixed point. The exception is the result of Wright and Bidwell [Wright and Bidwell, 1997], who show stable cycling behavior corresponding to very "weird" mutation and crossover distributions that would never be used in practice. Cycling behavior has also been shown in biological population genetics models [Hastings, 1981].

The random heuristic search model also leads in a natural way to a Markov chain model where the states are (finite) populations. Vose has a number of results that connect the infinite population model to the finite population model in the limit as the population goes to infinity. These theorems assume that the heuristic $\mathcal{G}$ is focused. For example, he shows that ([Vose, 1999], theorem 13.1), the probability of being in a given neighborhood of the set of fixed points can be made arbitrarily high by choosing the population size to be sufficiently large.

Thus, there is a lot of interest in knowing whether the heuristic that defines the infinite population model of a genetic algorithm is focused. This paper gives numerical examples where the infinite population model of a genetic algorithm exhibits stable cycling/chaotic behavior, which implies that the heuristic is not focused.

We expect that the examples of this paper very well could arise in practice if the mutation and selection described in this paper was used. However, they are not examples of the simple genetic algorithm in that a density-dependent mutation scheme is used.

Chaotic behavior could also be useful for restoring diversity in a run of a genetic algorithm that is not making progress. When the GA seems to have converged, the parameters could be adjusted to introduce chaotic behavior, which can move the algorithm from a local optimum.

We are aware of one other paper that discusses chaos (more accurately fractals) and genetic algorithms. Juliany and Vose [Juliany and Vose, 1994] generated fractals by determining the basins of attractions of fixed points of $\mathcal{G}$

1 Notation

Let Ω be the search space of length ℓ binary strings, and let $n = 2^\ell$. For $u, v \in \Omega$, let $u \otimes v$ denote the bitwise-and of u and v, and let $u \oplus v$ denote the bitwise-xor of u and v. Let $\overline{u}$ denote the ones-complement of u, and $\#u$ denote the number of ones in the binary representation of u. Let $\mathbf{1}$ denote the string of ones (or the integer $2^\ell - 1$). Thus, $\overline{u} = \mathbf{1} \oplus u$.

Integers in the interval $[0, n) = [0, 2^\ell)$ are identified with the elements of Ω through their binary representation. If $j \in \Omega$, we assume that j_0 denotes the least significant bit of the binary representation of j and $j_{\ell-1}$ denotes the most significant bit. However, when we write j as a binary string, we will use conventional notation with the least significant bit on the right.

This correspondence allows Ω to be regarded as the product group

$$\Omega = Z_2 \times \ldots \times Z_2$$

where the group operation is $\oplus$. The elements of Ω corresponding to the integers 2^i, $i = 0, \ldots, \ell - 1$ form a natural basis for Ω.

A population for a genetic algorithm over length ℓ binary strings is usually interpreted as a multiset (set with repetitions) of elements of Ω. A population can also be interpreted as a 2^ℓ dimensional incidence vector over the index set Ω: if x is a population vector, where x_i is the proportion of the element $i \in \Omega$ in the population. This implies that $\sum_j x_j = 1$. For example, suppose that $\ell = 2$ so that Ω is identified with the set $\{0, 1, 2, 3\}$. Then the population $\{0, 0, 2, 2, 3\}$ is represented by the population vector p where $p_0 = 2/5$, $p_1 = 0$, $p_2 = 2/5$, and $p_3 = 1/5$. We would also write $p = \langle 2/5, 0, 2/5, 1/5\rangle^T$.

Let

$$\Lambda = \{p \in R^n : \sum_i p_i = 1 \text{ and } x_i \geq 0 \text{ for all } i \in \Omega\}.$$

Thus any population vector is an element of Λ. Geometrically, Λ can be interpreted as the $n - 1$ dimensional unit simplex in R^n. Note that elements of Λ can be interpreted as probability distributions over Ω.

If $expr$ is a Boolean expression, then

$$[expr] = \begin{cases} 1 & \text{if } expr \text{ is true} \\ 0 & \text{if } expr \text{ is false} \end{cases}$$

If p is a population, $i \in \{0, \ldots, \ell - 1\}$, and $k \in \{0, 1\}$, let

$$S(p, i, k) = \sum_{j \in \Omega} p_j [j_i = k].$$

In other words, $S(p, i, k)$ is the relative frequency of population elements whose ith bit has the value k. For example, if $\ell = 2$ and if $p = \langle 2/5, 0, 2/5, 1/5\rangle^T$ is the example population described above, then $S(p, 0, 0) = p_0 + p_2 = 4/5$, $S(p, 0, 1) = p_1 + p_3 = 1/5$, $S(p, 1, 0) = p_0 + p_1 = 2/5$, and $S(p, 1, 1) = p_2 + p_3 = 3/5$. $S(p, i, k)$ can be also interpreted as the "schema average" of the schema with one fixed position in position i where the value of that fixed position is k.

In the *random heuristic search* model, a population-based generational search algorithm over Ω is defined by a heuristic function $\mathcal{G} : \Lambda \to \Lambda$. Given a population of size r which is represented by $p \in \Lambda$, the next generation population is obtained by taking r independent samples from the probability distribution $\mathcal{G}(p)$. When the simple genetic algorithm is modeled by random heuristic search, the heuristic function $\mathcal{G}$ can be represented as the composition of a selection heuristic function $\mathcal{F}$, and a mixing heuristic function $\mathcal{M}$. See [Vose, 1999] for more details. In this paper, we express $\mathcal{G}$ as $\mathcal{G}(p) = \mathcal{F} \circ \mathcal{M}(p)$, rather than the more usual $\mathcal{G}(p) = \mathcal{M} \circ \mathcal{F}(p)$.

2 Density-dependent mutation

One of the major practical difficulties in the practical use of genetic algorithms is "premature convergence". The genetic algorithm population loses diversity before sufficient exploration is done to discover the solutions of interest. A number of techniques have been proposed to prevent or slow down premature convergence. These include crowding [DeJong, 1975], sharing [Goldberg and Richardson, 1987], and partial reinitialization [Eshelman, 1991].

In this section, we propose another method to avoid premature convergence, based on maintaining population diversity. The idea is to use a different mutation rate at each string position, and this rate depends on the convergence of the population at that string position. If a string position is highly converged in a population, then a high mutation rate will be used at that string position in the production of the next generation. The theoretical justification of this method can be found in [Leung et al., 1997]; building on the concept of *degree of population diversity*, the authors show that premature convergence on a chromosome location (that is, the probability for allele loss on that position) decreases with population size and increases with $|m - 1/2|$ where m is the mutation rate. As we choose to keep the population size fixed, the suggestion is straightforward: Use population diversity as a quantitative mea-

sure to prevent premature convergence by adaptively varying mutation probability. This also corresponds to the *complementing schema in case of stagnation* procedure introduced by [Agapie and Dediu, 1996] for solving *deceptive* problems.

When there is a mutation rate at each string position, then the process of mutation of a chromosome is to mutate the loci of the chromosome independently. In other words, the probability of flipping the bit at string position i is the mutation rate m_i at that string position.

Next, we show how this adaptive mutation rate can be expressed in the framework of the infinite population model.

Mutation can be described in terms of a probability distribution over mutation masks. If $j \in \Omega$ is a binary string and $u \in \Omega$ is a mutation mask, then the result of mutating j using u is $j \oplus u$. Since mutation masks are elements of the search space, a probability distribution over mutation masks is an element of Λ. Given such a probability distribution $\mu \in \Lambda$, the corresponding mutation heuristic is defined by

$$\mathcal{U}(p)_k = \sum_{u\in\Omega}\sum_{j\in\Omega} \mu_u p_j [u \oplus j = k] = \sum_{j\in\Omega} \mu_{k\oplus j} p_j.$$

In the case where there is a mutation rate m_i for each string position,

$$\mu_u = \prod_{i=0}^{\ell-1} (1 - u_i - m_i + 2u_i m_i)$$

where u_i denotes the bit value of u at string position i, $i = 0, 1, \ldots, \ell - 1$. For example, if $\ell = 5$ and $u = 01001$, then $\mu_u = (1 - m_0)m_1(1 - m_2)(1 - m_3)m_4$.

Under density-dependent mutation, the string position i mutation rate m_i is a function of the current population. In particular, it is a function of the population bit frequencies $S(p, i, 0)$ and $S(p, i, 1)$ at position i. A one way to define such a function is to define a function $r : [0, 1] \to [0, 1/2]$ with the property that $r(1 - x) = r(x)$, and where r has a minimum at $x = 1/2$ and maxima at $x = 0, 1$. Then we define

$$m_i = r(S(p, i, 0)) = r(S(p, i, 1)).$$

One particular family of such functions is defined by:

$$r_{a,b}(x) = 2^{a-1} b \left| x - \frac{1}{2} \right|^a \tag{1}$$

where $a \geq 1$ and $0 \leq b \leq 1$. Under this definition, $r_{a,b}(0) = r_{a,b}(1) = b/2$ and $r_{a,b}(1/2) = 0$. Thus, m_i is $b/2$ when position i has completely converged in the population, and m_i is zero when the bit values at position i have equal frequency.

Figure 1 shows a graph of $r_{4,1}$.

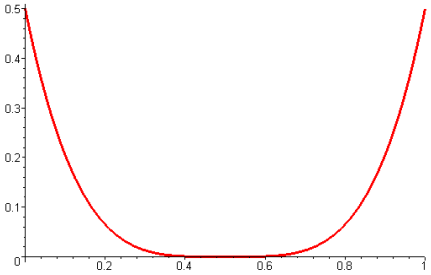


Figure 1: Mutation-adaptation rule: mutation increases whenever diversity is low on a chromosome position.

3 Truncation selection

In this section we show how truncation selection can be modeled by random heuristic search. Denote the population size by r. Under classic truncation selection, a number t with $0 < t < r$ is given. The the current population is ordered by fitness, and the t most fit individuals are selected for reproduction. For simplicity, we assume that all individuals in the population have distinct fitnesses. The most fit t individuals, when represented as a population vector in Λ, define a probability distribution. Under our adaptation of truncation selection to the random heuristic search model, the population at the next step is formed by taking r independent samples from this probability distribution.

This procedure can be defined as a selection heuristic. Without loss of generality, we assume that Ω is ordered so that $f_0 < f_1 < \ldots < f_{n-1}$, where f_j denotes the fitness of $j \in \Omega$. Let $T = t/r$. Then the truncation selection heuristic function $\mathcal{F}_T : \Lambda \to \Lambda$ is defined by:

$$\mathcal{F}_T(p)_k = \tag{2}$$

$$\begin{cases} 0 & \text{if } T < \sum_{k<j} p_j \\ \frac{T - \sum_{k<j} p_j}{T} & \text{if } \sum_{k<j} p_j \leq T < \sum_{k\leq j} p_j \\ \frac{p_k}{T} & \text{if } \sum_{k\leq j} p_j \leq T. \end{cases} \tag{3}$$

It can be verified that this formula agrees with the above procedure for every finite population size. An example will be given in the next section.

4 The 1-bit GA case

We first show how to obtain cycling and chaos in the dynamical system model when the genetic algorithm representation has only a single bit. In this case, the model is 1-dimensional since a population $p = \langle p_0, p_1 \rangle^T$ can be completely described by p_1 since $p_0 = 1 - p_1$. Thus, we identify Λ with $[0, 1]$ under the correspondence $\langle p_0, p_1 \rangle \longleftrightarrow p_1$. Our objective is to describe a heuristic function $\mathcal{G} : [0, 1] \to [0, 1]$ which describes one generation of the GA so that the dynamical system determined by $\mathcal{G}$ exhibits stable cycling.

In the 1-bit case, the truncation selection heuristic defined in the previous section reduces to the following:

$$\mathcal{F}_T(p_1)_1 = \begin{cases} 1 & \text{if } T < p_1 \\ \frac{p_1}{T} & \text{if } p_1 \leq T. \end{cases}$$

In the 1-bit case, the mutation heuristic is:

$$\mathcal{U}(p_0, p_1)_0 = (1-m)p_0 + mp_1$$
$$\mathcal{U}(p_0, p_1)_1 = mp_0 + (1-m)p_1$$

where m is the mutation rate.

Reducing this to the one variable $p = p_1$ gives

$$\mathcal{U}(p) = m - (2m-1)p = m(1-2p) + p$$

If the $r_{a,b}$ function is used for density-dependent mutation, then

$$\begin{aligned} \mathcal{U}_{a,b}(p) &= 2^{a-1}b\left|p - \frac{1}{2}\right|^a (1-2p) + p \\ &= p - \frac{b}{2}|2p-1|^a(2p-1) \end{aligned}$$

It can be verified that $\mathcal{U}_{a,b}$ is continuously differentiable even when $a = 1$ and thus $r_{a,b}$ is not continuously differentiable.

Since there is no crossover possible in the 1-bit case, the heuristic that defines the 1-bit dynamical system describing the genetic algorithm is $\mathcal{G}_{a,b,T} = \mathcal{F}_T \circ \mathcal{U}_{a,b}$.

Figure 2 shows graphs of the selection heuristic $\mathcal{F}_T$ for $T = 7/10$ and the mutation heuristic $\mathcal{U}_{a,b}$ for $a = 2$ and $b = 1$.

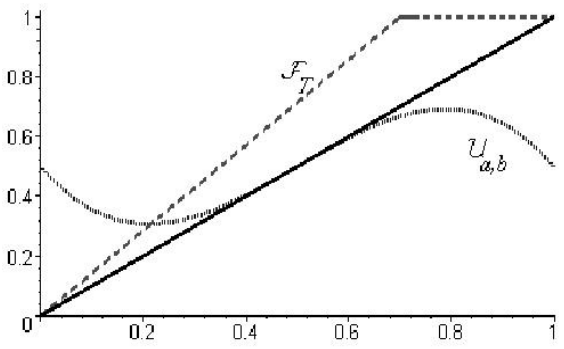


Figure 2: Heuristics $\mathcal{F}_T$ (selection) for $T = 7/10$, res. $\mathcal{U}_{a,b}$ (mutation) for $a = 2$ and $b = 1$

Note that under our assumption that $f_0 < f_1$, if the maximum value of the mutation heuristic function $\mathcal{U}$ is less than T, then the region where the selection heuristic has value 1 is never reached, and the GA heuristic $\mathcal{G}_{a,b,T}$ is continuously differentiable.

We show later that the behavior of $\mathcal{G}$ undergoes a phase transition as $\mathcal{G}$ goes from being continuously differentiable to having a discontinuous derivative. Thus, it is of interest to determine the conditions on a, b, and T that assure that $\mathcal{G}$ is continuously differentiable.

Lemma 1 *If T satisfies the condition*

$$T \geq \frac{1}{2}\left(\frac{1}{b(a+1)}\right)^{\frac{1}{a}}\left(\frac{a}{a+1}\right) + \frac{1}{2}$$

then $\mathcal{G}$ is continuously differentiable.

Proof.

As above, $\mathcal{G}$ is continuously differentiable if and only if the maximum value of $\mathcal{U}$ is less than or equal to T. This maximum value will occur when $p > 1/2$, so we can drop the absolute value from the the formula that defines $\mathcal{U}$. Thus,

$$\mathcal{U}(p) = p - \frac{b}{2}(2p-1)^{a+1}$$

Now we solve the equation $\frac{\partial U}{\partial p} = 0$.

$$\begin{aligned} \frac{\partial U}{\partial p}(p_0) = 0 \quad &\Longleftrightarrow \quad (2p_0 - 1)^a = \frac{1}{b(a+1)} \\ &\Longleftrightarrow \quad p_0 = \frac{1}{2}\left(\frac{1}{b(a+1)}\right)^{\frac{1}{a}} + \frac{1}{2} \end{aligned}$$

Then we substitute p_0 into U to obtain the maximum value of $\mathcal{U}$.

$$\begin{aligned} U(p_0) &= \frac{1}{2}\left(\frac{1}{b(a+1)}\right)^{\frac{1}{a}} + \frac{1}{2} - \frac{b}{2}\left(\frac{1}{b(a+1)}\right)^{\frac{a+1}{a}} \\ &= \frac{1}{2}\left(\frac{1}{b(a+1)}\right)^{\frac{1}{a}}\left(\frac{a}{a+1}\right) + \frac{1}{2} \end{aligned}$$

This value gives the minimum for T such that $\mathcal{G}$ is continuously differentiable. □

Figure 3 shows the area of (a, T) space where $\mathcal{G}$ is continuously differentiable.

A fixed point z of a 1-dimensional continuously differentiable discrete-time dynamical system $y \longrightarrow g(y)$ is stable if $|g'(z)| < 1$ and is unstable if $|g'(z)| > 1$. The system is *focused* if f is continuously differentiable and if $\lim_{t\to\infty} g^t(y)$ converges for every starting point y. If $\lim_{t\to\infty} g^t(y) = z$ and if g is continuous, then z is a fixed point of g.

We can now give a more rigorous justification of stable cyclic behavior of the dynamical system defined by $\mathcal{G}$ for some specific values of the parameters.

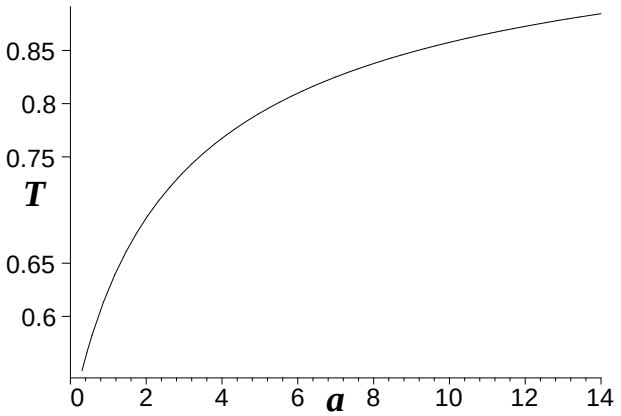


Figure 3: $\mathcal{G}$ is continuously differentiable in the region above the curve.

For the above example with $a = 2$, $b = 1$, and $T = 7/10$, the dynamical system defined by $y \longrightarrow \mathcal{G}(y)$ exhibits stable cycling. $\mathcal{G}$ has a single fixed point at $z = 0.908430$, and $\mathcal{G}'(z) = -1.431114512$, so the fixed point is unstable. $\mathcal{G}^2 = \mathcal{G} \circ \mathcal{G}$ has three fixed points at 0.756838, 0.908430, and 0.984383, and the derivative of $\mathcal{G}^2$ at these points are -0.7721976, 2.0480887, and -0.7721976. Thus, $\mathcal{G}^2$ has two stable fixed points that map to each other under $\mathcal{G}$. This demonstrates that $\mathcal{G}$ exhibits stable cycling of period 2. Figure 4 shows the graphs of the identity function, $\mathcal{G}$, and $\mathcal{G}^2$ for $a = 2$, $b = 1$, and $T = 7/10$.

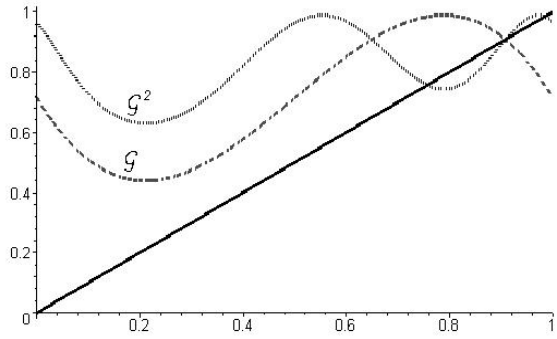


Figure 4: Heuristics $\mathcal{G}$ and $\mathcal{G}^2$ for $a = 2$, $b = 1$, $T = 7/10$

5 Empirical Results Using the Infinite Population Model

We implemented the 1-bit and multi-bit infinite population models in the Maple™programming language. Maple™, along with other mathematical symbolic processing languages, allows for both symbolic processing and arbitrary precision floating point computation.

To assure correctness of the code, we implemented the models in several different ways (1-bit, multi-bit, with and without the Walsh transform), and we cross-checked the results of the different implementations.

5.1 The 1-bit case

For the 1-bit case, we produced what [Peitgen et al., 1992] call *final-state* or *Feignebaum* diagrams. We used the following algorithm for a fixed value of the parameters a, T and b.

1. Choose an initial value p_0 at random from the interval $[0, 1]$.
2. Carry out 100 iterations to compute $p_1, p_2, \ldots, p_{100}$ using $p_{n+1} = \mathcal{G}(p_n)$.
3. Carry out 200 more iterations to compute $p_{101}, \ldots, p_{300}$.
4. Plot $p_{101}, \ldots, p_{300}$ on the diagram.

To produce a diagram, we fixed one of the a and T parameters and varied the other. The b parameter was fixed at 1. Figure 5 shows the diagram with T fixed at $7/8$ and a varying from 5 to 14 with an increment of 0.01. Figure 6 shows the diagram with a fixed at 4 and T varying from 0.6 to 0.99 in increments of 0.001. Both diagrams show period-doubling approach to chaos as the a or T parameter approaches the boundary of the region where $\mathcal{G}$ is continuous. In the region where $\mathcal{G}$ is discontinuous, the behavior seems to be mostly periodic, but with some chaotic deviation from the period behavior for some parameter values.

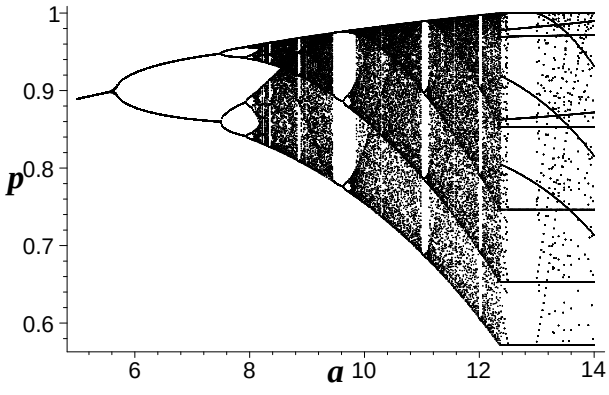


Figure 5: Final-state diagram for $\mathcal{G}$ heuristic for $T = 7/8$, $b = 1$, and values of a from 5 to 14

These diagrams were generated with 100 digits of floating-point precision. However, the diagrams look identical to

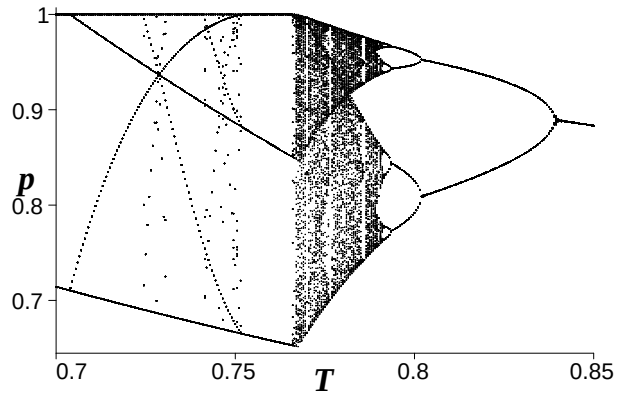


Figure 6: Final-state diagram for $\mathcal{G}$ heuristic for $a = 4$, $b = 1$, and values of T from 0.7 to 0.9

diagrams produced using 10 digits of floating-point precision. Lack of precision means that specific iterates may not be computed correctly (due to the sensitivity of initial conditions due to chaos), but the overall behavior is not affected.

5.2 The multibit case

We also did a number of runs using 2-bit to 4-bit representations, and using one-point and uniform crossover with crossover rates from 0 to 1. The fitness function used assigned one plus the integer value of the binary representation of a string to that string. Thus, the fitness of the 3-bit string 000 was 1, the fitness of 001 was 2, etc.

We found that the behavior of the multi-bit representation models were qualitatively the same (e. g., same period of cycling) as the 1-bit model for the same values of a and T. The presence or absence of crossover and the crossover rate did not affect the results. To be more specific, we ran the model for the combinations of the a and T parameters that are given table 1 for bit lengths of 2 to 4, for crossover rate 1/2, and for both one-point and uniform crossover. To check for cycling of period C, we ran for 100 iterations with a random initial population, and then looked at the 1-norm of the difference between the final population and the population C iterations prior to the final iteration. To check for chaos, we ran both an initial random population and a small random perturbation of this initial random population for 100 iterations, and looked at the 1-norm of the difference between the final populations. The cycling checks were run with 50 decimal digit floating point precision and the chaos checks were run with 100 digit floating point precision.

In the check for cycling, the maximum deviation (1-norm) between the last population and the population C iterations back was 1.3×10^{-6}, and in the check for chaos, a perturbation of size approximately 10^{-50} grew to a difference of at least 10^{-38} after 100 iterations.

$T\backslash a$	4	6	8	10
3/4	10	3	3	3
5/6	2	Chaos	4	4
7/8	1	2	8	Chaos
9/10	1	1	2	4
11/12	1	1	1	2
13/14	1	1	1	1

Table 1: Cycle length or chaotic behavior for 1-bit to 4-bit representations, for different values of the parameters a and T

We did some experiments with the multibit model where the parameters for the bits were set separately. When one bit was set to give cycling behavior and the other bits had a constant mutation rate, the model exhibited cyclic behavior of the same period as when all bits were set to give cycling behavior. And when one bit was set to give chaotic behavior and the remaining bits had constant mutation, the model exhibited chaotic behavior. When bits were set to give cyclic behavior of different periods where the shorter periods were divisors of the longest period, and where $\mathcal{G}$ was continuously differentiable, the longest period resulted. When one bit was set with parameter values that did not make $\mathcal{G}$ continuously differentiable and corresponded to period 3, and other bits were set to give period 10, period 3 resulted.

6 Empirical Results Using A Finite Population GA

We implemented the finite population genetic algorithm that corresponds to the infinite population model described above. To show how the behavior depends on the population size, we did many runs of the case where $r = 5/6$ and $a = 4$. As shown in table 1, the infinite population model has cyclic behavior of period 2 in this case. As a test for cyclic behavior, we looked at when the population average fitness exhibited period 2 cyclic behavior. Let f_t denote the average population fitness at time t. We would say that the GA has cyclic behavior at time t (over 4 generations) if either $f_{t-3} \leq f_{t-2}$, $f_{t-2} \geq f_{t-1}$, and $f_{t-1} \leq f_t$; or if $f_{t-3} \geq f_{t-2}$, $f_{t-2} \leq f_{t-1}$, and $f_{t-1} \geq f_t$. Note that the probability of this happening for a given value of t for a sequence of uniformly distributed random numbers is $1/4$.

Table 6 shows the results of running the GA for 100 runs for 1020 generations with population sizes 100, 250, 1000, and 5000. The fitness function was the same as used for the infinite population model. The string length was 50, and 1-point crossover with a crossover rate of 1/2 was used.

Truncation selection was used with $r = 5/6$, and bitwise density dependent mutation with $a = 4$ was used. The table shows how many of the last 1000 generations exhibited cyclic behavior as defined above.

These results show that the infinite population model makes predictions about a finite population genetic algorithm that can be verified with a population size of 100.

Population size	100	250	1000	5000
# generations	660.8	804.4	968.0	1000.0
Standard error	18.1	16.5	8.7	0.0

Table 2: Number of generations exhibiting cyclic behavior out of 1000

7 Conclusion

We have shown that introducing a bitwise density dependent mutatation in conjunction with truncation selection into a bit-representation genetic algorithm can cause the infinite-population model of this genetic algorithm to exhibit cyclic and chaotic behavior. As the mutation parameter increases, the model goes through a period-doubling approach to chaos.

This work is significant for two reasons.

First, it shows that the very important Vose infinite population model can exhibit a qualitatively different kind of behavior, namely chaos, than has been seen before.

Second, it demonstrates a new way to introduce diversity into an evolutionary computation algorithm, namely the cyclic and chaotic behavior shown in this paper. To follow up on this, more work needs to be done on characterizing the conditions under which cyclic and chaotic behavior can occur.

Acknowledgements

This work was done while the second author was visiting the University of Montana, supported by a COBASE grant from the National Research Council, USA.

References

[Agapie and Dediu, 1996] Agapie, A. and Dediu, H. (1996). GA for deceptive problems: Inverting schemata by a statistical approach. In *Proceedings IEEE International Conf. on Evolutionary Computation (ICEC'96)*, pages 336–340, Nagoya, Japan. IEEE.

[DeJong, 1975] DeJong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI.

[Eshelman, 1991] Eshelman, L. (1991). The CHC adaptive search algorithm: how to have safe search while engaging in nontraditional genetic recombination. In Rawlings, G. J. E., editor, *Foundations of genetic algorithms*, pages 265–283, San Mateo. Morgan Kaufmann.

[Goldberg and Richardson, 1987] Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, Hillsdale, N. J. Lawrence Erlbaum Associates.

[Hastings, 1981] Hastings, A. (1981). Stable cycing in discrete-time genetic models. *Proc. Nat. Acad. Sci. USA*, 78:7224–7225.

[Juliany and Vose, 1994] Juliany, J. and Vose, M. D. (1994). The genetic algorithm fractal. *Evolutionary Computation*, 2(2):165–180.

[Leung et al., 1997] Leung, Y., Gao, Y., and Xu, Z. B. (1997). Degree of population diversity - a perspective on premature convergence in genetic algorithms and its markov chain analysis. *IEEE Trans. on Neural Networks*, 8:1165–1176.

[Peitgen et al., 1992] Peitgen, H.-O., J:urgens, H., and Saupe, D. (1992). *Chaos and Fractals, New Frontiers of Science*. Springer-Verlag, New York.

[Vose, 1999] Vose, M. D. (1999). *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA.

[Vose and Liepins, 1991] Vose, M. D. and Liepins, G. E. (1991). Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44.

[Vose and Wright, 1994] Vose, M. D. and Wright, A. H. (1994). Simple genetic algorithms with linear fitness. *Evolutionary Computation*, 4(2):347–368.

[Wright and Bidwell, 1997] Wright, A. H. and Bidwell, G. L. (1997). A search for counterexamples to two conjectures on the simple genetic algorithm. In *Foundations of genetic algorithms 4*, pages 73–84, San Mateo. Morgan Kaufmann.

Genetic Algorithms: A Fitness Formulation for Constrained Minimization

J. A. Wright
Department of Civil and Building Eng.
Loughborough University,
Loughborough,
Leicestershire, LE11 3TU, UK
J.A.Wright@lboro.ac.uk

R. Farmani
Department of Civil and Building Eng.
Loughborough University,
Loughborough,
Leicestershire, LE11 3TU, UK
R.Farmani@lboro.ac.uk

Abstract

A fitness formulation is presented for solving constrained optimization problems. In this method, the dimensionality of the problem is reduced by representing the constraint violations by a single infeasibility measure. The infeasibility measure is used to form a two stage penalty that is applied to the infeasible solutions. The performance of the method has been examined by its application to a set of eleven test cases. The results have been compared with previously published results from literature. It is shown that the method is able to find the optimum solutions. The proposed method is easy to implement and requires no parameters. The approach is also robust in its handling of both linear and nonlinear equality and inequality constraint functions. Furthermore, the method does not require an initial feasible solution.

1 INTRODUCTION

In the last two decades, genetic algorithms have received much attention regarding their potential as global optimization techniques. More recently, the solution of constrained optimization problems has been addressed by some researchers. However, the methods developed still have several limitations and there is no single technique which could overcome all the problems posed by constrained optimization. In what follows, the merits and limitations of the different constraint handling methods is discussed.

1.1 Constraint Handling Methods for Numerical Optimization Problems

Penalty function methods are the most common methods in handling constraint optimization problems. In these methods, a penalty term is added to the objective function for the degree of violation of constraints (static penalty) or the degree of violation of constraints as well as the generation number (dynamic penalty) (Homaifar et al, 1994, Joines and Houck, 1994). The death penalty method simply rejects infeasible individuals so it is necessary to initialize a population with feasible solution. In general the weakness of penalty methods is that they require a large number of parameters (to adjust the relative weights of each constraint in the penalty, and the weight of the penalty against the objective function). Penalty approaches also fail to capitalize on any information generated by the search on the nature of the solution space.

Michalewicz and Janikow (1991), presented the GENOCOP method which is based on designing specialized operators that incorporate knowledge of the constraints. This method uses projection operators that map feasible points back to feasible boundaries. Their system only handles linear constraints with any objective function with a feasible starting point and effectively reduces search space. Schoenauer and Michalewicz (1996), constructed operators which maintain solution on non-linear analytical constraint surfaces.

In order to avoid generating and rejecting a large number of infeasible solutions, specialized operators can be used. In the Greedy decoder method, the chromosome does not directly encode a solution in the feasible region but rather a set of parameters is used by the decoder to generate a feasible solution. Because the decoder must be guaranteed to never produce infeasible solutions, it is often extremely difficult to design. Hajela and Yoo (1995), overcame this problem in an

alternative approach that is able to handle both non-linear, equality and inequality constraints. Their approach derives from the fact that the structure of both feasible and infeasible solutions is present in the population at any generation of the search. The basis for this scheme is that those segments of the binary chromosome (bit string), that contribute to calculating the objective function are minimally altered, while those segments of the chromosome that contribute to constraint violations are replaced by corresponding segments from feasible chromosomes.

Schoenauer and Xanthakis (1993), presented behavioral a memory method which considers the problem constraints in a sequence; a switch from one constraint to another is made upon arrival of a sufficient number of feasible individuals in the population. The success of the whole process is highly dependent on the genetic diversity maintained during the initial steps, thus ensuring a uniform sampling of the feasible region.

Constrained multi-objective methods use the value of the objective function and the penalty values, or the number of constraint violations, as elements of a vector and apply multi-objective techniques to minimize all components of the vector. Surry, et al. (1997), presented the COMOGA method in which all members of the population are ranked on the basis of the constraint violations. Such a rank, together with the value of the objective function, lead to a two-objective optimization problem. Cheng and Li (1997), presented another constrained multi-objective optimization methodology. The approach integrates a Pareto genetic algorithm and fuzzy penalty function. In this method, the rank of a solution is determined by knowing the solutions status (feasible or infeasible), the distance from the Pareto optimal set, and position in infeasible region. The fuzzy-logic penalty function method, having a discrete membership function, can express the rank order of solutions in a Pareto optimization and transform a multi-objective constrained optimization into an unconstrained problem.

Hybrid methods combine evolutionary techniques with deterministic optimization procedures for numerical optimization problems. Myung et. al, (1996), presented a two phase evolutionary programming method based on a hybrid method. During the first phase, an evolutionary algorithm is used to optimize the function. In the second phase of the optimization, Lagrange multipliers are used to place emphasize on the violated constraints whenever the best solution does not fulfill the constraints. By updating the Lagrange multipliers, the trial solutions are driven to the optimal point where all constraints are satisfied.

Koziel and Michalewicz (1999), presented the homomorphous mapping approach for solving constrained optimization problems. The method incorporates a homomorphous mapping between an n-dimensional cube and the feasible search space. The method introduces an additional problem-dependent parameter to partition the interval [0,1] into subintervals of equal length such that the equation of each constraint has at most one solution in every subinterval. The method loses the locality feature of the mapping for non-convex feasible search spaces, and a small change in the coded solution may result in large change in the solution itself. The method requires additional computational effort for finding all the intersection points for a line segment with the boundaries of the feasible region.

Runarsson and Yao (2000), introduced a stochastic ranking method in which the objective function values are used for ranking the solutions in the infeasible region of the search space. A probability parameter is used to determine the likelyhood of two individuals in the infeasible space being compared to each other. Although the method proved to be effective in solving a wide range of constrained optimization problems, it was also sensitive to the choice of probability parameter.

Most of these constraint handling methods are problem dependent. They often require user supplied parameters to be adjusted in order to obtain good performance from the method. Some of the methods are also only able to handle only specific constraint types and therefore lack generality. Some of the methods limit the search to the feasible search space. However, a good search should approach the optimum solution from both sides of the feasible/infeasible border (Richardson et al, 1989). Only two methods appears to be able to solve many types constrained optimization problem, the homomorphous method (Koziel and Michalewicz, 1999) and the stochastic ranking method (Runarsson and Yao, 2000). The disadvantages of the homomorphous method are that it requires an initial feasible solution and that all infeasible solutions are rejected. Another limitation is the need for problem-dependent parameters in the method. One possible restriction on the stochastic ranking method is that it is sensitive to the form of the underlying evolutionary algorithm.

2 THE FITNESS FORMULATION

The fitness formulation described here addresses the limitations of existing constraint handling methods. In particular, it is independent of any parameters and can be used without an initial feasible solution being given.

The approach is also robust in its handling of both linear and nonlinear equality and inequality constraint functions. The methodology described here applies to the minimization of an objective function $f(\mathbf{X})$, subject to inequality constraints $g_j(\mathbf{X})$, and equality constraints $h_j(\mathbf{X})$, (Equations 1, 2 and 3):

$$f(\mathbf{X}) = f(x_1, ..., x_n) \tag{1}$$

and

$$g_j(\mathbf{X}) \leq 0, \quad (j = 1, ..., q) \tag{2}$$

$$h_j(\mathbf{X}) = 0, \quad (j = q+1, ..., m) \tag{3}$$

The method has been formulated to ensure that slightly infeasible solutions with a low objective function value remain fit. This is seen as a benefit to solving highly constrained problems that have solutions on one or more of the constraint bounds. In contrast, solutions furthest from the constraint bounds are seen as containing little genetic information that is of use and are therefore penalized.

This is achieved through the application of a two part penalty function. The first penalty ensures that the worst of the infeasible solutions has an objective function value that is higher or equal to that of the best solution in the population. The second penalty increases the objective function value of the infeasible solutions in proportion to their infeasibility. The approach is implemented in three stages, first an infeasibility is assigned to each individual, second the "best" and "worst" individuals in the population are identified, and finally the two part penalty function is applied to the infeasible solutions.

2.1 Chromosome Infeasibility

The infeasibility of an individual should represent both the number of active constraints and the extent to which each constraint is violated. A measure of infeasibility that has these properties is the sum of the constraint values for all violated constraints. This can be evaluated in two stages. First the feasible constraint values are reset as zero and infeasible values as positive (Equation 4); a small tolerance δ is applied to the equality constraints to aid the finding of a feasible solution. Second, the solutions infeasibility ($\imath(\mathbf{X})$), is taken as the sum of the normalized constraint values (Equation 5).

$$c_j(\mathbf{X}) = \begin{cases} \max\left(0, g_j(\mathbf{X})\right), & \text{if } 1 \leq j \leq q \\ \max\left(0, \left(|h_j(\mathbf{X})| - \delta\right)\right), & \text{if } q+1 \leq j \leq m \end{cases} \tag{4}$$

$$\imath(\mathbf{X}) = \frac{\sum_{j=1}^{m} \frac{c_j(\mathbf{X})}{c_{max,j}}}{m} \tag{5}$$

The constraint violation values are normalized since large differences in the magnitude of the constraint values can lead to dominance of the infeasibility by constraints having the highest values. The scaling factor for each constraint $c_{max,j}$, is taken as the maximum value of the constraint violation in the current population. Resetting the scaling factor for each population provides a further dynamic element to the infeasibility calculation. This has been found to give better algorithm performance than for a static scaling factor; for instance, by basing $c_{max,j}$ on the constraint violations in the first population.

2.2 Identification of the Bounding Solutions

The penalties applied to the infeasible solutions are a function of the infeasibility and objective function value for the "best" and "worst" individuals in the population. For a population containing at least one or more feasible solution, the "best" individual $\check{\mathbf{X}}$, is the feasible solution having the lowest objective function value. If all individuals are infeasible however, then the best solution is taken as the solution having the lowest infeasibility (regardless of the objective function value of the individuals).

The "worst" individual $\hat{\mathbf{X}}$, is selected by comparing all individuals against the best individual ($\check{\mathbf{X}}$). Two potential population distributions exist in relation to this comparison.

- The first population distribution occurs when **one or more** of the infeasible solutions has an objective function value that is **lower** than the "best" solution. In this case, the "worst" individual is taken as the infeasible solution having the highest infeasibility and an objective function value that is lower than the "best" solution's. If more than one individual exists with the same highest infeasibility, then the "worst" individual is taken as the solution with maximum infeasibility and the lower of the objective function values.

- The second population distribution occurs when **all** of the infeasible solutions have an objective function value that is **greater** than the "best" solution. Here the "worst" individual is identified as being the solution with the highest infeasibility. If more than one individual exists with the same highest infeasibility, then the "worst" individual

is taken as the solution with maximum infeasibility and the higher of the objective function values.

2.3 Chromosome Fitness

Since we are concerned with the minimization of the objective function, the infeasible solutions are been penalized prior to the conversion of the objective function values to fitness form. The conversion to fitness ($F(\mathbf{X})$), is by the simple subtraction of the penalized objective function values ($\ddot{f}(\mathbf{X})$), from the maximum penalized value in the current population. The objective function values of the infeasible solutions are penalized according to the solutions infeasibility in relation to that of the "worst" solution ($\imath(\hat{\mathbf{X}})$) and the "best" solution ($\imath(\check{\mathbf{X}})$). Note that if a feasible solution exists, then the "best" solution is feasible and will have a zero infeasibility ($\imath(\check{\mathbf{X}})$=0.0).

The penalty is applied in two stages. The first stage only applies if one or more infeasible solutions has a lower and therefore potentially better objective function value than the "best" solution ($(f(\mathbf{X}) < f(\check{\mathbf{X}})) \wedge (\imath(\mathbf{X}) > 0.0)$). If this relationship holds, then the penalty is applied to all of the infeasible solutions; if the relationship does not hold, then the first penalty is not applied to any solution. The goal of the first penalty is to increase the objective function value of the infeasible solutions such that the "worst" solution has an objective function value that is equal to that of the "best" solution. This has been implement using a simple linear relationship between the objective function values and the infeasibility of the "best" and "worst" solutions (Equations 6 and 7).

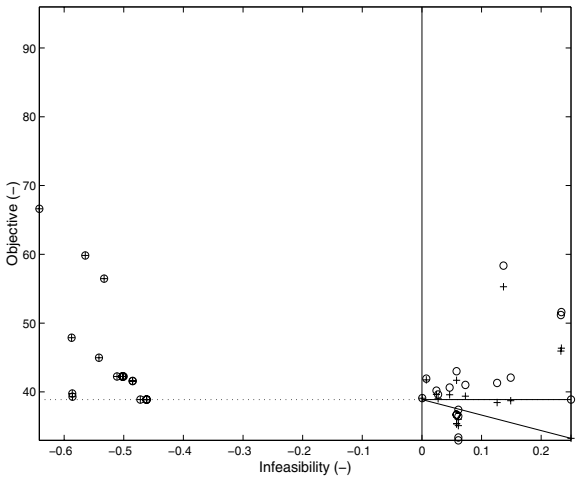
$$\tilde{\imath}(\mathbf{X}) = \frac{\imath(\mathbf{X}) - \imath(\check{\mathbf{X}})}{\imath(\hat{\mathbf{X}}) - \imath(\check{\mathbf{X}})} \tag{6}$$

$$\dot{f}(\mathbf{X}) = f(\mathbf{X}) + \tilde{\imath}(\mathbf{X})\ (f(\check{\mathbf{X}}) - f(\hat{\mathbf{X}})) \tag{7}$$

Note that if the penalty is not applied the $\dot{f}(\mathbf{X}) = f(\mathbf{X})$. The application of the first penalty is illustrated by Figure 1 in which the original solutions are indicated by a "+" and the penalized solutions by a "o". The "best" and "worst" solutions are connected by a line (with the infeasibility of the "best" solution being reset to zero). Note that in this and subsequent figures, a negative infeasibility indicates a feasible solution (the negative infeasibilities only being assigned for the purposes of illustrating the distribution of solutions in the current population).

The second penalty exponentially increases the objective function values such that the objective function of

Figure 1: Application of the First Penalty



the "worst" individual is twice that of the best (Equations 8 and 9). A penalty equal to one "best" objective function value has been found to give good performance of the algorithm for a range of problems and is therefore considered here to be a constant. Similarly, an exponential weighting parameter of 2.0 has been found to give good performance over a range of problems and is therefore also considered to be a constant. The exponential function gives a slight reduction in the rate of penalty applied to solutions of low infeasibility, thus helping to maintain the fitness of the slightly violated solutions.

$$\ddot{f}(\mathbf{X}) = \dot{f}(\mathbf{X}) + \gamma \left|\dot{f}(\mathbf{X})\right| \left(\frac{\exp(2.0\ \tilde{\imath}(\mathbf{X})) - 1.0}{\exp(2.0) - 1.0}\right) \tag{8}$$

$$\gamma = \begin{cases} 1.0, & \text{if}(f(\hat{\mathbf{X}}) \leq f(\check{\mathbf{X}})) \\ 0.0, & \text{if}(f(\hat{\mathbf{X}}) \geq (f(\check{\mathbf{X}}) + \left|f(\check{\mathbf{X}})\right|)) \\ \frac{f(\check{\mathbf{X}}) + \left|f(\check{\mathbf{X}})\right| - f(\hat{\mathbf{X}})}{f(\check{\mathbf{X}})}, & \text{otherwise} \end{cases} \tag{9}$$

The scaling factor γ, simply ensures that the maximum penalty is equivalent to the value of the "best" individual's objective function value. The second case in Equation (9) ($\gamma = 0.0$), applies when the "worst" individual already has a value that is greater than this amount. Here, no penalty is applied since the infeasible solutions would naturally have a low fitness and should not be penalized further. The absolute values of the objective functions in Equations (8) and (9) are necessary to allow the minimization of objective functions having a negative value.

The application of the second penalty is illustrated

by Figure 2 (the first penalized objective values being indicated by "o" and the second penalized objective values by "×").

Figure 2: Application of the Second Penalty

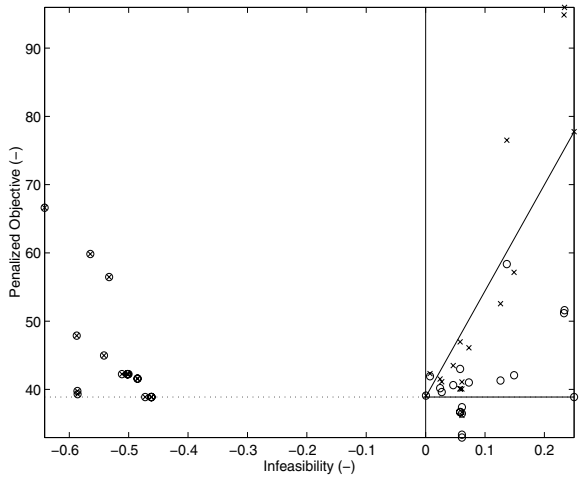
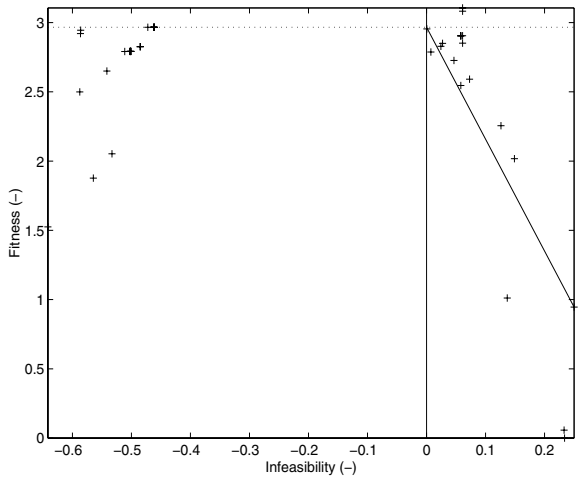


Figure 3, shows the penalized objective function values converted to fitness form. Note that the fittest individuals lie in both the infeasible and feasible regions. This allows the slightly infeasible, low objective function value solutions to be selected for reproduction. Note also that in this case, the fittest individual is infeasible.

Figure 3: Solution Fitness



It is evident that the approach is dynamic in the allocation of the penalty in that the absolute value of the penalty depends on the value of the "best" individual's objective function. The penalty also accounts for the range of infeasibility in the current population and the distribution of the infeasible solutions in relation to the "best" individual in the population.

3 TEST CASES

The performance of the proposed constraint handling method has been evaluated using a set of eleven test cases (Koziel and Michalewicz, 1999; Michalewicz and Fogel, 2000). These test cases include various forms of objective function (linear, quadratic, cubic, polynomial, nonlinear), each test case also having a different number of variables (n). The test problems also pose a range of constraint types and number of constraints (linear inequalities, LI; nonlinear equalities, NE; and nonlinear inequalities, NI). The general form of each test case is given in Table 1, which also indicates the number of constraints active at the optimum solution (a).

Table 1: Summary of Test Cases

Function	n	Form of $f(\mathbf{X})$	LI	NE	NI	a
G1	13	quadratic	9	0	0	6
G2	k	nonlinear	0	0	6	1
G3	k	polynomial	0	1	0	1
G4	5	quadratic	0	0	6	2
G5	4	cubic	2	3	0	3
G6	2	cubic	0	0	2	2
G7	10	quadratic	3	0	5	6
G8	2	nonlinear	0	0	2	0
G9	7	polynomial	0	0	4	2
G10	8	linear	3	0	3	6
G11	2	quadratic	0	1	0	1

The fitness formulation described here has been implemented and evaluated using simple genetic algorithm with binary encoding of the variables (25 bits used represent each variable). The implementation uses a "roulette wheel" selection strategy, single point crossover, mutation by changing bit values, and finally an elitist replacement strategy.

The performance of the method has been compared directly to the results obtained from the homomorphous mapping method (Koziel and Michalewicz, 1999), and therefore, where possible the same genetic algorithm (GA), parameter values have been adopted (a population size of 70; 90% probability of crossover; and a probability of mutation between 0.3% and 0.5%, Koziel and Michalewicz use a variable probability rate). Although Runarsson and Yao (2000), give results for the same test problems, a direct comparison of the algorithm performance to this method is not possible since they perform different experiments to those given in this paper, (particularly in terms of the number of function evaluations). For each test case, the three types of experiment described by Koziel and Michalewicz (1999), have been performed:

- Experiment 1; 20 runs each starting from a different randomly generated population; the maximum number of generations was set to 5,000.
- Experiment 2; the same as experiment 1, except that the maximum number of generations was increased to 20,000.
- Experiment 3; was also carried out in the same way as experiment 1, except that all runs started with a feasible individual (which was taken as the best solution obtained in experiment 1); however, only 10 different initial generations were optimized.

Table 2 shows the known optimal solution for each problem. The result of experiments 1 and 2 and 3 are given in Tables 3, 4 and 5 respectively. Tables 6, 7, and 8 give the corresponding results reported by Koziel and Michalewicz (1999).

Table 2: Summary of Optimum Values

Function	Optimum value
G1	-15
G2	0.803553
G3	1.0
G4	-30665.5
G5	5126.4981
G6	-6961.8
G7	24.306
G8	0.095825
G9	680.63
G10	7049.33
G11	0.75

A comparison of the two sets of results indicates that the method described here can find a more optimal solution than the method reported by Koziel and Michalewicz (1999); these solutions are shown in "**bold**" in Tables 3, 4, and 5. This is particularly the case for Experiment 3. For Experiments 1 and 2, it could be argued that the two approaches are comparable. However, the method by Koziel and Michalewicz (1999) requires an initial feasible solution, whereas the approach described here has no requirement for an initial feasible solution and can begin with a completely infeasible population. The ability to find a feasible solution as well as the optimum solution represents a significant improvement in algorithm performance.

The ability to find a feasible solution was examined in Experiment 1. For 8 of the 11 test cases (G1, G2, G3, G4, G6, G8, G9, G11), the algorithm described here found a feasible solution for all 20 independent runs (and in many cases, the search also found the global optimum). Furthermore, on average, the first feasible solution for this group of problems problems was found within 19 generations, (a maximum of 1,330 function evaluations). However, problems G7 and G10 required a higher number of generations in order to find a feasible solution (on average, 41 generations for G7 and 285 generations for G10). Further, feasible solutions G10 were only found for 18 of the 20 runs, and in 19 of 20 runs for G7. Feasible solutions for G5 were only found for 2 of the 20 runs.

The use of fixed parameters also proved to be effective, although for problem G4, improved results were obtained by reducing the weight of the second penalty (indicated as "small γ", in Tables 3, 4, and 5). A comparison of the results from Experiments 1 and 3, also makes it clear that the method is not dependent on the selected reference point for finding global optimum. However, reference point selection had effect on average scores.

4 CONCLUSIONS

This paper introduces a fitness formulation for constraint minimization. The infeasibility values are represented by the sum of the normalized constraint violation values. The infeasibility measure has the properties that it increases in value with both the number of active constraints and the magnitude of each constraint violation. The infeasibility measure is used to form a two stage dynamic "penalty" which applied to the infeasible solutions. The penalty is applied such that the slightly infeasible solution having a low objective function value are allowed to remain fit. It is shown, that this approach gives comparable, if not improved results than existing methods. The principle advantages of the approach are that first, it does not require any parameters, and second, that it is able to find the global optimum starting with a completely infeasible population of solutions. The method is also able to solve a range of constrained optimization problems, having both non-linear equality and inequality constraints.

Acknowledgements

The authors acknowledge the UK, Engineering and Physical Science Research Council for funding this work.

Table 3: Results from Experiment 1

Function	Experiment #1		
	worst	best	average
G1	-12.9519	**-14.9996**	**-14.84**
G2	0.7205	0.79434	0.76739
G3	0.99027	**0.99937**	**0.99812**
G4	-30261.6	-30624.1	-30547.915
G4(small γ)	-30403.7	-30661.12	-30568.65
G5		**5126.64487**	
G6	**-6347.8**	**-6948.85**	**-6484.06**
G7	**37.98319**	**24.672**	31.52044
G8	0.0267	0.09588	**0.089135**
G9	712.869	681.5615	688.05
G10	10572.66	7298.136	**8776.7699**
G11	0.9884	0.75	0.8151

Table 4: Results from Experiment 2

Function	Experiment #2		
	worst	best	average
G1	**-14.9081**	**-14.9996**	**-14.988**
G2	0.77091	0.79664	0.78465
G3	**0.99807**	0.9994	**0.99902**
G4	-30604.2	**-30650.01**	-30609.97
G4(small γ)	-30403.7	**-30661.1**	-30591.765
G5	**5135.4409**	**5126.6398**	**5131.0404**
G6	**-6347.8**	**-6961.37**	**-6657.81**
G7	37.9273	24.6707	30.927
G8	**0.06413**	0.09588	**0.09246**
G9	689.6234	681.1982	684.413
G10	10343.04	7152.832	8255.847
G11	0.90296	0.75	0.808

Table 5: Results from Experiment 3

Function	Experiment #3		
	worst	best	average
G1	-14.9691	**-14.9996**	-14.9911
G2	0.7812	**0.8003**	0.7845
G3	**0.9989**	**0.9994**	**0.9991**
G4	-30604.2	-30650	-30609.025
G4(small γ)	-30604.5	-30661.1	-30615.809
G5		**5126.63997**	
G6	-6347.8	**-6958.44**	-6692.61
G7	30.723	**24.61897**	27.3009
G8	0.09314	0.09588	0.095328
G9	685.7403	**681.162**	683.4643
G10	8357.847	**7289.063**	7788.533
G11	0.795	0.75	0.768

Table 6: Results from Experiment 1 (Koziel and Michalewicz, 1999)

Function	Experiment #1		
	worst	best	average
G1	-14.0566	-14.7207	-14.4609
G2	0.78427	0.79506	0.79176
G3	0.9917	0.9983	0.9965
G4	-30617.0	-30662.5	-30643.8
G5			
G6	-4236.7	-6901.5	-6191.2
G7	38.682	25.132	26.619
G8	0.0291434	0.095825	0.0871551
G9	682.88	681.43	682.18
G10	11894.5	7215.8	9141.7
G11	0.75	0.75	0.75

Table 7: Results from Experiment 2 (Koziel and Michalewicz, 1999)

Function	Experiment #2		
	worst	best	average
G1	-14.6154	-14.7864	-14.7082
G2	0.79119	0.79953	0.79671
G3	0.9978	0.9997	0.9989
G4	-30643.8	-30645.9	-30655.3
G5			
G6	-5473.9	-6952.1	-6342.6
G7	25.069	24.62	24.826
G8	0.0291438	0.095825	0.0891568
G9	683.18	680.91	681.16
G10	9659.3	7147.9	8163.6
G11	0.75	0.75	0.75

Table 8: Results from Experiment 3 (Koziel and Michalewicz, 1999)

Function	Experiment #3		
	worst	best	average
G1	-14.5732	-14.7184	-14.6478
G2	0.78279	0.79486	0.78722
G3	0.996	0.9978	0.997
G4	-30645.6	-30661.5	-30653.1
G5			
G6	-6390.6	-6944.4	-6720.4
G7	26.182	25.09	25.545
G8	0.0958246	0.095825	0.0958248
G9	683.58	681.72	682.56
G10	7685.8	7321.2	7498.6
G11	0.75	0.75	0.75

References

A. Homaifar, C. X. Qi, and S. H. Lai (1994). Constrained Optimisation Via Genetic Algorithms. *Simulation*, **62**(4), 242-254.

J. A. Joines, and C. R. Houck (1994). On the Use of Non-Stationary Penalty Functions to Solve Non-linear Constrained Optimisation Problems with GA's. *IEEE Conference on Evolutionary Computation*, **2**, 579-584.

Z. Michalewicz, and C. Z. Janikow (1991). Handling Constraints in Genetic Algorithms. *Proceedings of the International Conference on Genetic Algorithms*, **4**, 151-157.

M. Schoenauer, and Z. Michalewicz (1996). Evolutionary Computation at the Edge of Feasibility. International Conference on Evolutionary Computation, *The 4th conference on Parallel Problem Solving from Nature*, **4**, 245-254. September 1996, Berlin, Germany

P. Hajela, and J. Yoo (1995). Constraint Handling in Genetic Search - A Comparative Study. *AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference - Collection of Technical Papers*, **4**, 2176-2186.

M. Schoenauer and S. Xanthakis (1993). Constrained GA Optimisation. *Proceedings of the International Conference on Genetic Algorithms*, **5**, 573-580.

P. D. Surry and N. J. Radcliffe (1997). The COMOGA Method: Constrained Optimisation by Multi-Objective Genetic Algorithms. *Control and Cybernetics*, **26**(3), 391-412.

F. Y. Cheng, and D. Li (1997). Multiobjective Optimisation Design with Pareto Genetic Algorithm. *Journal of Structural Engineering*, **123**(9), 1252-1261.

H. Myung, and J. H. Kim (1996). Constrained Optimisation Using Two-Phase Evolutionary Programming. *Proceedings of IEEE International conference on Evolutionary Computation*, 262-267.

S. Koziel, and Z. Michalewicz (1999). Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimisation. *Evolutionary computation*, **7**(1), 19-44.

J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. *Proceedings of the International Conference on Genetic Algorithms*, **3**, 191-197.

Z. Michalewicz, and D. B. Fogel (2000). *How to Solve It: Modern Heuristics.* Berlin: Springer-Verlag.

T. P. Runarsson and X. Yao (2000). Stochastic Ranking for Constrained Evolutionary Optimisation. *IEEE Transactions on Evolutionary Computation*, **4**(3), 284-294.

An incremental fitness function for partitioning parallel tasks

Annie S. Wu, Shiyuan Jin, Guy Schiavone
School of EECS
University of Central Florida
Orlando, FL 32816-2362
aswu@cs.ucf.edu, sjin@ist.ucf.edu, guy@pegasus.cc.ucf.edu

Kuo-Chi Lin
Institute for Simulation and Training
University of Central Florida
Orlando, FL 32826-0544
klin@pegasus.cc.ucf.edu

Abstract

We describe a novel GA approach to partition programs to be executed on a parallel system. Two unique features distinguish this GA from traditional GA programs. First, this GA uses a dynamically incremental fitness function which starts out rewarding for simpler goals, gradually increasing the difficulty of the desired fitness values or goals until a full solution is found. Second, this GA uses a flexible representation style which allows the GA itself more control over both the structure and the value of the evolved solutions.

1 Introduction

We successfully apply a genetic algorithm (GA) to the problem of partitioning a collection of tasks to be executed on a group of parallel processors.

Parallel programming improves the performance of a program by breaking the program down into smaller tasks and executing these subtasks simultaneously where possible. Executing the subtasks of a program in parallel can significantly speed up the run time of a program. The quality of a parallel algorithm can be evaluated with the following equation:

$$Speedup = \frac{Run\_time\_of\_best\_sequential\_algorithm}{Run\_time\_of\_parallel\_algorithm}.$$

Obviously, the larger the speedup, the better the parallel algorithm. Ideally, one hopes to achieve a maximum speedup of P when solving a problem using P parallel processors. In practice, such a speedup cannot be achieved for every problem for the following reasons:

- It is not always possible to decompose a problem into P tasks, each requiring $1/P$ of the total sequential time to execute.
- The structure of the parallel computer may impose restrictions that render the optimum running time unattainable (e.g. synchronization and communication overhead).

Many factors can affect the exploitation of parallelism in programs, including the partitioning of programs, the balancing of computational load among processors, and the overhead created by data communication between processors. The problem of partitioning a parallel program involves the specification of sequential units of a program that can be executed concurrently by the available processors such that the total parallel computation time of the program is minimized when inter-processor communication costs are included. The efficient partitioning of a program into parallel subtasks is not a trivial problem. (Bokhari 1981) has shown that there does not exist a polynomial algorithm to solve this problem.

Given the number of possible orderings of tasks in processors, the percentage of valid orderings can be very small. If a GA is not restricted to only work with valid individuals, the chance of randomly finding a valid ordering, let alone a good valid ordering, may be very low. Restricting a GA to only form valid individuals, however, may introduce unexpected biases in the system. In addition, such a strategy may require extensive revision of the system with each new problem.

We describe here a novel GA approach which can successfully partition programs to be executed on a parallel system. Our GA places no restrictions on the individuals that can be formed and does not require special operators or repair mechanisms to ensure validity. Rather, it attempts to give partial fitness for invalid individuals that contain some valid subsequences

of tasks and encourages the formation of successively longer valid subsequences. Two unique features distinguish this GA from traditional GA programs. First, this GA uses a flexible representation style which allows the GA itself more control over both the structure and the value of the evolved solutions. Second, this GA uses a dynamically incremental fitness function which starts out rewarding for simpler goals, gradually increasing the difficulty of the desired fitness values or goals until a full solutions is found.

2 Related work

A variety of heuristic algorithms have been proposed to solve the partitioning problem. These algorithms include mapping-based methods (Sadayappan and Ercal 1987; Sadayappan, Ercal, and Ramanujam 1990) and clustering-based methods (Kim and Browne 1988; Sarkar 1989; Maheshwari and Shen 1998)

Researchers have also applied GAs to the problem of partitioning parallel programs. Hou et al. (1994) evolve strings which represent processor task schedules. Their approach restricts the actions of genetic operators to ensure the validity of evolved individuals. As a result, some parts of the search space may be unreachable. Correa et al. (1999) improve upon Hou's method to allow the entire search space to be searched. Tsuchiya et al. (1998) propose a GA based scheduler which incorporates the idea of task duplication: one task may be assigned to multiple processors. They compare their GA approach to a heuristic scheduling algorithm called Duplication Scheduling Heuristic (DSH) and show that the GA is able to find better solutions. All of these GA approaches require special methods to ensure the validity of the initial population and to ensure the validity of offspring generated by crossover and mutation. In other words, all individuals generated by these systems must represent "executable" partitions.

3 Algorithm design

We have implemented a novel GA approach for partitioning programs to be executed on parallel systems. We extend the traditional GA (Holland 1975; Goldberg, Korb, and Deb 1989) in two ways. First, we use a flexible representation style which allows the GA to dynamically evolve the structure and value of the solutions. Second, we use an dynamically adaptive, incremental fitness function which initially rewards for simple goals and gradually increases the difficulty of the goals over the generations. Our GA places no restrictions on the individuals that can be formed; both valid and invalid individuals may occur.

```
(3,0)(1,3)(2,2)(1,2)(3,0)(4,3)(5,2)(0,0)(2,1)
```

Figure 1: An example individual.

3.1 Problem representation

Previous work indicates benefits in using location independent problem representations (Goldberg, Korb, and Deb 1989; Holland 1975; Wu and Lindsay 1996) where the information content is not dependent on its location on a GA individual. Such a representation combined with genetic operators that rearrange and exchange information is expected to allow a GA to find building blocks (learn how to arrange related or epistatic information close together). Compactly arranged building blocks (building blocks with low defining length) are expected to be more likely to be transmitted as a whole by the genetic operators during a reproduction event.

Each individual in our GA population consists of a vector of cells. We define a *cell* to be a task and processor pair: (t, p). Each cell indicates that task t is assigned to be processed on processor p. The number of cells in an individual may vary, so individuals in our GA population will vary in length. Figure 1 shows an example individual. The first cell of this individual assigns task 3 to processor 0, the next cell assigns task 1 to processor 3, etc. This representation requires that the number of processors and number of tasks to be performed are known in advance. The problem itself defines the number of tasks to be performed and their dependencies on each other. We assume that the number of available processors is also defined in advance.

The cells on an individual determine which tasks are assigned to which processors. The order in which the cells appear on an individual determines the order in which the tasks will be performed on each processor. Individuals are read from left to right to determine the ordering of tasks on each processor. For example, the individual shown in figure 1 results in the processor assignments and ordering of tasks shown in figure 2. Invalid task orderings will have their fitness value penalized by the fitness function.

The same task may be assigned more than once to different processors. The example individual in figure 1 assigns tasks 1 and 2 twice. Tasks may not be assigned to the same processor more than once. Should a task-processor pair appear more than once on an individual, only the first (leftmost, since individuals are read

Processor 0	Task 3	Task 0	
Processor 1	Task 2		
Processor 2	Task 2	Task 1	Task 5
Processor 3	Task 1	Task 4	

Figure 2: Assignment of tasks from individual in figure 1.

from left to right) pair is active. Any remaining identical pairs are essentially non-coding regions. In the example from figure 1, the second instance of `(3,0)` is not scheduled into the processor lists in figure 2.

As each (task, processor) pair is read from left to right, all active pairs are placed into FIFO queues based on their processor specification. The content of each queue indicates the tasks that will be performed on each processor. The order of the tasks in each queue indicates the order in which the tasks are assigned to be executed on each processor. Thus, the order in which tasks will be performed on each processor depends on the order in which the task-processor pairs appear on an individual.

3.2 Genetic operators

The novel representation requires some modification in the genetic operators used. We use both crossover and mutation in our algorithm. The modified versions of these genetic operators are described here.

3.2.1 Crossover

Recall that each individual consists of a vector of task-processor pairs or cells. Crossover exchanges substrings of cells between two individuals. This allows the GA to explore new solutions while still retaining parts of previously discovered solutions.

Initially we will use random one-point crossover. Random crossover involves two parent individuals. This operator randomly selects a crossover point on each parent and exchanges the segments to the left of the crossover points to form two offspring. Figure 3 shows an example of random crossover.

The crossover rate parameter gives the probability that a pair of parents will undergo crossover. Parents that do not crossover undergo only mutation to form offspring.

3.2.2 Mutation

The mutation rate indicates the probability that a cell will be changed. As a result, the expected number of mutations per individual is equal to the mutation rate multiplied by the length of an individual. If a cell is selected to be mutated, then either the task number or the processor number of that cell will be randomly changed.

3.3 Fitness function

For this problem, a number of factors are expected to contribute to the fitness of an individual ("goodness" of a solution). These factors include but are not limited to:

1. Are all tasks performed? Does an individual contain at least one copy of each task?
2. Are the tasks scheduled on the processors in valid orders?
3. How long will it take the parallel processors to complete all tasks?
4. How well are the tasks distributed among the available processors.

Our task, then, is to determine a fitness function that effectively evaluates and incorporates each of the above components. The current fitness function breaks down the fitness function into two parts. The first part of the fitness function, $task\_fitness$, deals with items one and two above. The second part of the fitness function $processor\_fitness$, deals with items three and four. The actual fitness, $fitness$, of a GA individual is a weighted sum of the above two partial fitness values.

3.3.1 Calculating *task_fitness*

The *task_fitness* component of the fitness function evaluates whether all tasks are represented and in valid order. Because of the complexity of the solutions, we develop an incremental fitness function that changes over time. We will initially start out with a less demanding fitness function, and gradually increase the difficulty of the fitness function over time as the individuals in the GA population improve (Lohn, Haith, Columbano, and Stassinopoulos 1999). As in positive reinforcement training, this strategy rewards for small steps toward the goal, to encourage the algorithm to find the complete goal. We initially reward for finding short valid sequences of tasks. Over time, we increase the length of the sequences that can be rewarded, encouraging the GA to find and maintain longer valid sequences. Eventually the valid sequences will be long enough that the individuals will represent full valid solutions. In addition to "leading the GA towards a goal", this strategy also makes it possible to assign fitness values to valid sub-schedules even if the full represented schedule is not valid. As a result, no special

Randomly select parent 1 crossover point: 2
Randomly select parent 2 crossover point: 4

Parent 1	`(3,0)(1,3)`	`(2,2)(1,2)(4,3)(0,0)(2,1)`
Parent 2	`(3,2)(2,2)(4,1)(2,3)`	`(1,3)(2,2)`

Random crossover produces

Offspring 1	`(3,0)(1,3)`	`(1,3)(2,2)`
Offspring 2	`(3,2)(2,2)(4,1)(2,3)`	`(2,2)(1,2)(4,3)(0,0)(2,1)`

Figure 3: Random one-point crossover randomly selects crossover points on each parent and exchanges the left segments to form offspring.

actions need to be taken to restrict the formation of offspring individuals and the full search space is accessible.

The $task\_fitness$ component of an individual's fitness is based two main components:

- The percentage of valid sequences of a given length on an individual.
- The percentage of the total number of tasks specified by an individual.

Initially our fitness function will reward for short sequences of valid tasks. A sequence of tasks is valid if the tasks in the sequence are arranged in a valid chronological order. When the average fitness of the GA population exceeds a threshold fitness, the length of the sequence for which the GA searches is increased, thus increasing the difficulty of the fitness function. We anticipate several advantages to this approach. First, previous work indicates that gradually increasing the difficulty of a GA fitness function can result in the formation of more complex solutions (Lohn, Haith, Columbano, and Stassinopoulos 1999). Second, a dynamically changing fitness function is expected reduce the likelihood of premature convergence to partial solutions. A good partial solution (valid subsequence) that does not improve will be worth less over time as the fitness function becomes more demanding.

Calculating raw fitness

The raw fitness of an individual reflects the percentage of sequences of a given length in an individual that are valid sequences. For example, processor 2 in Figure 2 has been assigned three tasks. If our current sequence length is two, processor 2 contains two sequences of length two. Processor 2 contains only one valid sequence of length 2, the sequence `Task1-Task5`. The sequence `Task2-Task1` is not a valid sequence because Task 2 cannot be executed before Task 1.

Assume that the problem to be solved involves P processors and T tasks. Evolution will occur in eras, $era = 0, 1, 2, ..., E-1$. Initially, $era = 0$. The maximum era count, $E \leq T/P$, is a user defined parameter value. The era counter, era, is increased when the average population fitness exceeds a user defined threshold, *thresh* and when the number of individuals with the current maximum possible fitness exceeds a user defined threshold, *thresh_maxfit*.

Let $numtasks(p), p = 0, ..., P-1$, indicate the number of tasks assigned to processor p. To calculate the raw fitness of a processor, we need to consider two things: the first $era+1$ (or fewer) tasks assigned to the processor, and all task sequences of length $era+2$. The first component is important because as era increases, the likelihood of processors containing fewer than $era+2$ tasks increases. We need to reinforce the GA for these shorter sequences in order for them to eventually build up to the measured sequence length. This reinforcement also encourages distribution of tasks among the available processors.

We will first determine the contribution of the first $era+1$ or fewer tasks in a processor. Let

$$subseq(p) = \begin{cases} 1 & \text{if } numtasks(p) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

and let

$$valseq(p) = \begin{cases} 1 & \text{if the first } era+1 \text{ or fewer tasks in processor } p \text{ are in valid order} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Equations 1 and 2 refer to individual processors. To calculate the contribution over all processors (the contribution for the entire individual), we let

$$Subseq = \sum_{p=1}^{P-1} subseq(p)$$

$$Valseq = \sum_{p=1}^{P-1} valseq(p).$$

We will next determine the contribution of all sequences of length $era + 2$ in a processor. Let

$$s(p) = \#\text{ sequences of length } era + 2 \text{ in processor } p \quad (3)$$

and let

$$v(p) = \#\text{ valid seq. of length } era + 2 \text{ in proc. } p. \quad (4)$$

Combining equations 3 and 4 to determine the contribution over all processors we let

$$S = \sum_{p=1}^{P-1} s(p) \qquad V = \sum_{p=1}^{P-1} v(p).$$

The $raw\_fitness$ for an individual is then calculated with the following equation

$$raw\_fitness = \frac{Valseq + V}{Subseq + S}. \quad (5)$$

Calculating the task ratio

In addition to encouraging the system to find valid sequences of tasks, we also want to encourage the system to include at least one copy of each task in each solution. We define the *task_ratio* to be the percentage of distinct tasks from the total tasks in the problem that are represented in an individual. The *task_ratio* is calculated with the following equation:

$$task\_ratio = \frac{\begin{array}{c}number\ of\ distinct\ tasks\\ specified\ on\ an\ individual\end{array}}{\begin{array}{c}total\ number\ of\ tasks\\ in\ the\ problem.\end{array}} \quad (6)$$

This factor penalizes solutions that do not contain at least one copy of every task. Once all tasks are represented in an individual, this penalty becomes null.

Effective fitness

The effective $task\_fitness$ of an individual is the product of equations 5 and 6.

$$task\_fitness = raw\_fitness * task\_ratio \quad (7)$$

This value makes up the first component of the fitness of a GA individual.

3.3.2 Calculating *processor_fitness*

To fully optimize the use of parallel processors, the GA must be able to distribute tasks among the available processors. The $processor\_fitness$ component of the fitness function addresses this issue.

Suppose t is the run time for a solution represented by an individual. Let $serial\_len$ equal the length of time required to complete all tasks serially on a single processor and let $super\_serial\_len = P * serial\_len$ where P equals the number of processors. Any reasonable solution should give $t \ll super\_serial\_len$, making $super\_serial\_len$ a safe but reasonable upper bound to solution run time. The goal of the GA is to minimize t. The $processor\_fitness$ first calculates the difference between $super\_serial\_len$ and t then calculates what proportion of $super\_serial\_len$ this difference represents:

$$processor\_fitness = \frac{super\_serial\_len - t}{super\_serial\_len}.$$

As a result, $processor\_fitness$ is inversely proportional to t. As the run time of a solution decreases, the amount that $processor\_fitness$ contributes to the individual's full fitness increases.

It is important to note that although the theoretical maximum value of *processor_fitness* is 1.0, in practice, this value can not be achieved. For *processor_fitness* to equal 1.0, the run time, t, of a solution would have to be zero. Since all tasks obviously require non-zero run time, t will never be zero for valid individuals.

3.3.3 Calculating *fitness*

The full fitness of an individual is a weighted sum of the $task\_fitness$ and $processor\_fitness$:

$$fitness = (1-b) * task\_fitness + b * processor\_fitness.$$

Our initial experiments use $b = 0.1$. The value of b may be adjusted in future runs.

Even though the theoretical maximum value of *fitness* is 1.0, this value will never be achieved in practice because the *processor_fitness* component of the fitness function can never reach 1.0 in practice.

The value of $fitness$ is returned to the GA by the fitness function as the fitness of an individual.

4 Experimental details

A parallel program can be represented by its data flow graph $G = (V, E)$. This graph is a node-labeled and edge-labeled directed acyclic precedence graph (APG), where V is a set of nodes that represent sequential processes and E is a set of directed edges that specify both precedence constraints and communication paths

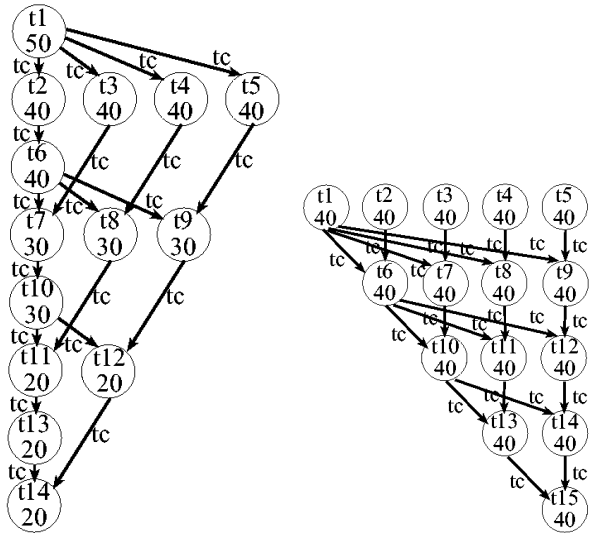


Figure 4: Description of task dependencies for problems C and B in (Tsuchiya, Osada, and Kikuno 1998).

among nodes. In an APG, each node gives the execution cost time and each edge represents the communication cost time required to pass data from one node to another during program execution.

We test our GA on several problems described by (Tsuchiya, Osada, and Kikuno 1998), specifically problems B and C from their paper. The APG for these problems are shown in figure 4. The communication time, *tc*, is set to 25 for problem B and 20 for problem C. For problem B, Tsuchiya et al. (1998) obtained a minimum run time of 300 with their GA and 315 with DSH. For problem C, they obtained a minimum run time of 260 with their GA (Tsuchiya 2001) and 260 with DSH.

The details of our GA are described in section 3. We used the following parameter settings in our experiments:

Population size	=	200
Crossover type	=	Random 1-point
Crossover rate	=	1.0
Mutation rate	=	0.005
Selection method	=	Tournament
Stopping condition	=	1500 generations.

Our GA performs 300,000 evaluations (= population size 200 × 1500 generations) in each run, making it comparable to the data from (Tsuchiya, Osada, and Kikuno 1998) (population size 30 × 10,000 generations = 300,000 evaluations). On average, however, our GA finds the best solution in about 600 generations, so significantly fewer evaluations are actually needed. We

Minimum run time

	$thresh = 0.7$	$thresh = 0.75$
$b = 0.1$	303 (6.32)	301.5 (4.47)
$b = 0.2$	304.5 (10.12)	302 (4.83)

Maximum fitness

	$thresh = 0.7$	$thresh = 0.75$
$b = 0.1$	0.9874 (0.0003)	0.9874 (0.0002)
$b = 0.2$	0.9746 (0.0008)	0.9748 (0.0003)

Figure 5: Results for problem B: average (and standard deviation) over ten runs.

tested the following variations in the fitness function parameters:

thresh	=	0.7, 0.75
b	=	0.1, 0.2
thresh_maxfit	=	0.1 .

For each experiment, we also include a set of runs in which the fitness function simply rewards for valid runs with minimal run time and does not use an incremental schedule. These runs are used as a comparison to judge the merits of our GA using an incremental fitness function.

5 Results

Our initial experiments test two values of *thresh* (which determines when the *era* counter will advance) and two values of *b* (which determines the contribution of the two main components of the fitness function). Ten runs were executed for each experiment. We report here the average performance over each set of ten runs.

Figure 5 gives the results for problem B. The top table in figure 5 shows the run time of the best solution (solution with minimum run time) found, averaged over ten runs. The minimum run time found is 300. This minimum value is achieved in 80% of the 40 runs represented in figure 5 and is comparable to the best GA solution found in (Tsuchiya, Osada, and Kikuno 1998). The bottom table in figure 5 shows the average maximum fitness found in the runs. While there is not a large difference in the average run times of each set of experiments, the data suggest that a lower value of *b* appears to produce solutions with higher fitness.

In addition to these experiments on problem B, we also run an experiment consisting of a set of ten baseline runs in which the fitness function simply rewarded for minimum valid run times. No incremental fitness is given in these runs. Seven of the ten runs are unable

Minimum run time

	$thresh = 0.7$	$thresh = 0.75$
$b = 0.1$	267 (9.49)	274 (13.50)
$b = 0.2$	275 (15.81)	271 (9.94)

Maximum fitness

	$thresh = 0.7$	$thresh = 0.75$
$b = 0.1$	0.9848 (0.0005)	0.9844 (0.0007)
$b = 0.2$	0.9687 (0.0018)	0.9692 (0.0011)

Figure 6: Results for problem C: average (and standard deviation) over ten runs.

to find any valid partitions within the 500 generations. Three find solutions with run times of 500, 435, and 655, all significantly higher than the run times found using the incremental fitness function. Closer inspection of these runs reveals that these solutions are not actually evolved. Rather they are randomly generated individuals that just happen to be valid individuals. They appear for only for a single generation and do not produce any viable offspring. In fact, all three of these solutions were randomly generated in the initial population and then immediately destroyed. No other valid solutions were found in any of these runs.

Figure 6 gives the results for problem C. The top table in figure 6 shows the run time of the best solution (solution with minimum run time) found, averaged over ten runs. The minimum run time found is 260, occurring in almost half of the 40 runs reported above. This value is comparable to the best GA and DSH solutions from (Tsuchiya 2001; Tsuchiya, Osada, and Kikuno 1998). The bottom table in figure 6 shows the average maximum fitness found in the runs. Again, a lower value of b appears to produce solutions with higher fitness. We also run a baseline experiment for problem C consisting of ten runs without incremental fitness. Eight of the ten runs are unable to find a valid solution within 500 generations. Two runs find solutions with a run time of 590 and 380. These solutions, again, appear to be random occurrences, the only valid individuals generated in these runs.

Figure 7 shows an example of how the population fitness of a run evolves in our GA. The top line shows the best population fitness at each generation. The bottom line shows the average population fitness at each generation. The vertical lines indicate the generations at which the *era* counter is incremented. The start of each era is indicated at the top of the graph. The average population fitness climbs within each era. Each time the *era* counter is incremented, however, the difficulty level of the fitness function increases and the

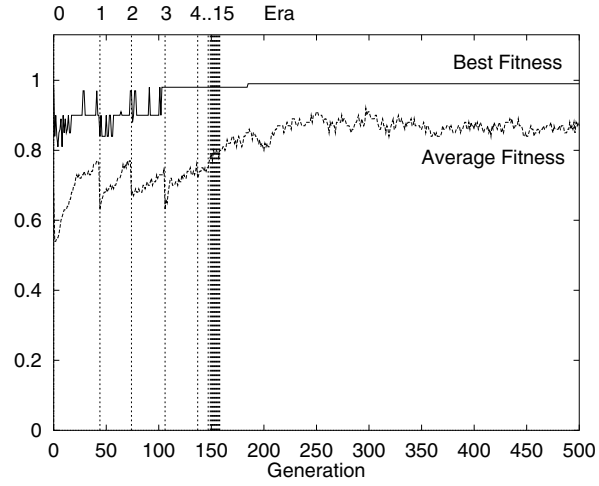


Figure 7: Evolution of population fitness in response to increasing eras.

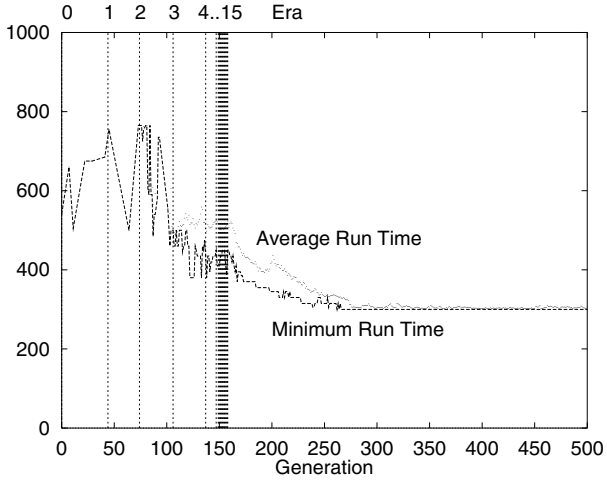


Figure 8: Evolution of run time in response to increasing eras.

average fitness of the population drops. After about six eras in this run, there are apparently enough valid task sequences to allow the remaining eras to increment once per generation until the maximum $era = 15$ is reached. Figure 8 shows the evolution of run time in the same run. The run times of the evolved solutions seem to be less reactive to the change in eras.

6 Conclusions and future work

We introduce a novel GA approach to evolving schedules that partition the tasks of programs to be executed on parallel processors. Our GA successfully solves this problem, finding solutions that are compa-

rable or better than solutions found by previous methods. The key advantage of our GA over previous GA approaches to this problem is that our GA does not use special operators or repair mechanisms to restrict the search to only valid regions of the search space. As a result, we eliminate biases that may occur due to the special operators, we know that all individuals within the search space are reachable, and we eliminate the need to re-tune or revise the special operators and repair mechanisms each time a new problem is introduced to be solved.

Two unique features distinguish our GA from traditional GA programs. First, this GA uses a dynamically incremental fitness function which starts out rewarding for simpler goals, gradually increasing the difficulty of the desired fitness values or goals until a full solutions is found. Second, this GA uses a flexible representation style which allows the GA itself more control over both the structure and the value of the evolved solutions.

It is clear from our baseline experiments that this is a difficult problem for a GA to solve if there are no restrictions placed on the evolutionary process. We modify the problem specific components of a GA – the problem representation and the fitness function – to encourage the GA to create and maintain valid individuals. The incremental fitness function initially rewards the GA for finding short valid sequences of tasks. As fitness improves, the lengths of these sequences increase, thus rewarding the GA for finding and keeping longer valid sequences. Valid individuals are further evaluated with respect to their quality so that shorter run times are rewarded.

We hope to further improve the performance of our GA with future developments. Issues that we plan to address include an analysis of the effectiveness of our representation format, an examination of the use of alternate genetic operators, and a study of the effects of variable genome length in a GA.

Acknowledgements

This research is supported in part by the Air Force Research Laboratories and Science Applications International Corporation. The authors would like to thank the anonymous reviewers for their helpful comments.

References

Bokhari, S. H. (1981). On the mapping problem. *IEEE Trans. Comput. 30*(3), 207–214.

Correa, R. C., A. Ferreira, and P. Rebreyend (1999). Scheduling multiprocessor tasks with genetic algorithms. *IEEE Trans. Parallel and Distrib. Syst. 10*(8), 825–837.

Goldberg, D. E., B. Korb, and K. Deb (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems 3*, 493–530.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hou, E. S., N. Ansari, and H. Ren (1994). A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel and Distrib. Syst. 5*(2), 113–120.

Kim, S. J. and J. C. Browne (1988). A general approach to mapping of parallel computations upon multiprocessor architectures. In *Proc. Int. Conf. Parallel Processing*, pp. 1–8.

Lohn, J. D., G. L. Haith, S. P. Columbano, and D. Stassinopoulos (1999). A comparison of dynamic fitness schedules for evolutionary design of amplifiers. In *Proc. 1st NASA/DoD Workshop of Evolvable Hardware*, pp. 87–92.

Maheshwari, P. and H. Shen (1998). An efficient clustering algorithm for partitioning parallel programs. *Parallel Computing 24*, 893–909.

Sadayappan, P. and F. Ercal (1987). Nearest-neighbor mapping of finite element graphs onto processor meshes. *IEEE Trans. Comput. 36*(12), 1408–1424.

Sadayappan, P., F. Ercal, and J. Ramanujam (1990). Cluster partitioning approaches to mapping parallel programs onto a hypercube. *Parallel Computing 13*, 1–16.

Sarkar, V. (1989). *Partitioning and scheduling parallel programs for multiprocessors*. MIT Press.

Tsuchiya, T. (2001). Personal communication: updated results.

Tsuchiya, T., T. Osada, and T. Kikuno (1998). Genetic-based multiprocessor scheduling using task duplication. *Microprocessors and Microsystems 22*, 197–207.

Wu, A. S. and R. K. Lindsay (1996). A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation 4*(2), 169–193.

On a New Constraint Handling Technique for Multi-Objective Genetic Algorithms

Jin Wu
Mechanical Engineering Dept.
University of Maryland
College Park, MD 20742

Shapour Azarm
Mechanical Engineering Dept.
University of Maryland
College Park, MD 20742

Abstract

A new constraint handling technique is developed to work with Multi-Objective Genetic Algorithms (MOGAs). This technique is based on a primary-secondary fitness assignment scheme, one that uses both individuals' fitness and matching. A Pareto ranking scheme is used for the primary fitness assignment wherein no subjective and problem dependent parameters are used. Rules that take the concept of "matching" into account are used for the secondary fitness assignment. Some new set quality metrics are introduced and used for a comparison of the new technique with a previous approach. Due to the stochastic nature of MOGA, confidence intervals with a 95% confidence level are obtained for the quality metrics based on the randomness in the initial population. An engineering example, namely the design of a vibrating platform, is used for the comparison and demonstration of the new technique.

1 INTRODUCTION

Engineering design optimization problems usually have a mix of continuous-discrete design variables, multiple objectives that are at least partly conflicting, and multiple constraints. The solutions to such optimization problems are known as Pareto solutions (Miettinen, 1999). Evolutionary algorithms such as Genetic Algorithms (or GAs) (Holland, 1975) are capable of solving single objective optimization problems with mixed variables. There have been different approaches to incorporate multiple objectives into GAs. Coello (1999) gives a comprehensive survey of evolutionary-based multi-objective optimization methods. Among these methods, a Multi-Objective Genetic Algorithm or MOGA (Fonseca and Fleming, 1993) is one of the methods that is capable of generating a Pareto solution set in a single run of the GA as opposed to solving a series of single objective optimization problems.

GAs, and thus MOGAs, are essentially unconstrained optimization techniques. Hence, the way that the constraints are handled in GAs or MOGAs becomes important. Most constraint handling techniques that reported in the literature for GA and/or MOGA focus on handling constraints during a fitness assignment stage. A fitness is used to interpret the mating performance, taking both objectives and constraints into account, and as an allocation of reproductive opportunities. For example, two individuals selected independently with high fitness values have a higher chance of producing better offsprings than those with low fitness values. Conventionally, only a single fitness is used throughout the procedure. The most common form of a single fitness assignment is to alter the fitness value of an individual by a penalty if it violates any constraint (Goldberg, 1989):

$$fitness_i = f_i(\boldsymbol{x}) + Q_i \quad (1)$$

where the quantity $fitness_i$ refers to the fitness of the i^{th} individual, $f_i(\boldsymbol{x})$ is the objective function value (to be minimized) of the i^{th} individual, and Q_i is a penalty function due to a constraint violation of the i^{th} individual. Since an aggregation between objectives and constraints is involved in Eq.(1), subjective and problem dependent penalty parameters are usually introduced in Q_i. Although, the use of a penalty method in GAs and MOGAs has been somewhat successful (Homaifar, et al., 1994; Michalewicz and Attia 1994; Narayana and Azarm, 1999; Kurapati et al., 2000; among others), the definition of a good penalty function or setting up the corresponding subjective and problem dependent parameters can be very critical and difficult. Among the recent constraint handling techniques, those that treat constraints as separate objectives and make use of a Pareto ranking scheme are the most promising ones. Surry et al. (1995) and Coello (2000) proposed methods that make use of Pareto rankings to handle constraints. Since these methods use Vector Evaluated Genetic Algorithms or VEGAs (Schaffer, 1985) to ensure solution feasibility, when the number of constraints increases, the required computational effort can increase dramatically. A comprehensive survey of these constraint handling

techniques can be found in Michalewicz (1995). So far, there have not been any published constraint handling techniques based on a Pareto ranking scheme applied to a MOGA.

In this paper, a new constraint handling approach is proposed for a MOGA. This new approach takes both individuals' performance and matching against one another into account by using a primary-secondary fitness assignment scheme. The primary fitness is used to measure individuals' performance, while the secondary fitness is used to interpret "matching" of two individuals. The objectives and constraints are handled via a Pareto ranking scheme in order to avoid their aggregation and the use of subjective and problem dependent parameters.

In order to show that the proposed constraint handling technique does work better with a MOGA, the constraint handling technique CH-I4 from Kurapati et al. (2000) is chosen for a comparison with the new constraint handling technique. In the literature, most of such comparisons are done in an ad-hoc manner (e.g., Azarm et al., 1999, Binh and Korn, 1997). Very few papers in the literature have reported on metrics for measuring and comparing the quality of observed Pareto solutions obtained from different evolutionary methods. Zitzler and Thiele (1998) proposed two metrics for the purpose of comparing four multiobjective evoluationary algorithms. Van Veldhuizen (1999) (wherein further references can be found) reviewed and defined nine metrics to assess the quality of Pareto solutions. More recently, five new set quality metrics are formulated in closed-forms and geometrically illustrated by Wu and Azarm (2000). In this paper, a few new set quality metrics are proposed and some metrics from Wu and Azarm (2000) are modified so that they can be used to compare multiple observed Pareto solution sets from different techniques in a more meaningful way. Confidence intervals with a 95% confidence level are used in the comparison study so that when analyzing the performance of the different techniques, their stochastic nature is accounted for.

The rest of the paper is organized as follows. The definition and terminology used in the paper are given in Section 2. The proposed constraint handling technique is discussed in Section 3. The details of the comparison study including the quality metrics and confidence intervals are described in Section 4. An engineering example, the design of a vibrating platform, is provided in Section 5 to demonstrate performance of the proposed technique. The paper is concluded with the remarks in Section 6.

2 DEFINITION AND TERMINOLGY

The formulation of a typical multi-objective design optimization problem with m objective functions is shown below in Eq.(2):

$$\begin{aligned} &\text{Minimize } f(x)=\{f_1(x), \ldots, f_i(x), \ldots, f_m(x)\} \\ &\text{subject to: } x\in D \\ &D = \{x\in \Re^n: g_j(x)\le 0,\ j=1, \ldots, J;\ h_k(x)=0,\ k=1, \ldots, K\} \end{aligned} \quad (2)$$

where x is a design vector containing n components of design variables, $f_i(x)$ is the i^{th} objective function, $g_j(x)$ is the j^{th} inequality constraint and $h_k(x)$ is the k^{th} equality constraint. The set of all design vectors that satisfies all constraints is denoted by D. The solution to a multi-objective optimization problem is a set of Pareto solutions: $X = (x_1, x_2, \ldots, x_{np})$, wherein for any point $x_j \in X$, there does not exist another point $x_k \in D$ with $k \neq j$, such that $f_l(x_k) \le f_l(x_j)$ for all $l = 1,\ldots,m$, with strict inequality for at least one l. In this paper, a Pareto solution set that truly meets this definition is also called a 'true' Pareto solution set or a Pareto frontier. In contrast, a Pareto solution set that is obtained by a multi-objective optimization method is referred to as an 'observed' Pareto solution set. In reality, an observed Pareto solution set is a finite set and is an estimate of the true Pareto solution set.

Let $P_1, P_2, \ldots, P_T$ denote the observed Pareto solution sets obtained from T different optimization runs or T different optimizers. For a solution $p_{j,k} \in P_k$, ($j=1,\ldots,np_k$, np_k is the number of Pareto solutions contained in the k^{th} observed Pareto solution set and $k=1,\ldots,T$), if there does not exist another solution $p_{l,s} \in P_1 \cup P_2 \cup \ldots \cup P_T$ ($l=1,\ldots,np_k$ and $s=1,\ldots,T$), with $j \neq l$, such that $f_i(x_{l,s}) \le f_i(x_{j,k})$, for all $i = 1,\ldots,m$ with strict inequality for at least one i, then $p_{j,k}$ is defined as a Meta Pareto solution. Total Meta Pareto set (TMP) is then defined as the best possible solutions obtained in $P_1, P_2, \ldots, P_T$, which means that $TMP=\{\ldots,p_{k,j},\ldots,p_{s,l},\ldots\}$ is the union of all the Meta Pareto solutions contained in $P_1, P_2, \ldots, P_T$. Figure 1 shows two observed Pareto solution sets and their total Meta Pareto solution set.

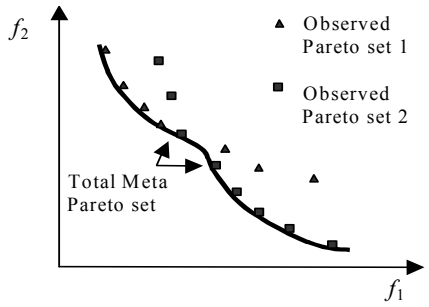


Figure 1: Total Meta Pareto Set

3 NEW CONSTRAINT HANDLING (NCH) APPROACH FOR MOGA

The proposed new constraint handling approach, as discussed in the following sections, uses a primary-secondary fitness assignment scheme to account for both individuals' performance and matching. A Pareto ranking approach is used for the primary fitness assignment so that subjective and problem dependent parameters are avoided. A rule based algorithm is developed for a secondary fitness assignment, at the selection stage, so that the "matching" between the individuals as parents is accounted for.

3.1 PRIMARY FITNESS ASSIGNMENT

Primary fitness expresses an individual's performance by taking both the objectives and constraints into account. Unlike the conventional penalty methods, no aggregation between objectives and constraints are involved in this Pareto ranking scheme.

The primary ranking has two stages. In the first stage, only the design objectives in the optimization problem are considered and all constraints are ignored. Dominant values of all individuals are calculated. The dominant value of an individual p_k is defined as the number of all other individuals in the population that dominate over p_k. At the second stage, the dominant value calculated from the first stage is taken as a new objective, and the sum of constraint violations is taken as the other objective. Dominant values of all individuals are calculated in this objective-constraint space again. Each individual is ranked based on its new dominant value. For those individuals with the same newly found dominant value, their rankings are modified based on their constraint violation. In this case, those individuals with a less amount of constraint violation are preferred over those with a more amount of constraint violation. Primary fitness can then be computed based on the ranks of all individuals. For a typical multi-objective optimization problem in the form of Eq. (2), a step-by-step approach for a primary fitness assignment is described next.

1) Consider: Minimize $f(x)=\{f_1(x),\ldots, f_m(x)\}$. These m objective functions are the m objective functions in Eq. (2). For an individual i (i =1,…,M), compute the dominant value $D_{obj.i,}$ in the objective space without considering any constraint. Here, M is the population size.
2) Compute the extent of constraint violation $C_{con,i}$ for an individual i (i =1,…,M), using Eqs. (3)-(5).

$$C_{con,i} = C^1_{con,i} + C^2_{con,i} \tag{3}$$

$$C^1_{con,i} = \sum_{j=1}^{J} \delta_{j,i} + \sum_{k=1}^{K} \delta_{k,i} \tag{4}$$

$\delta_{j,i}$ (or $\delta_{k,i}$) = 1 if g_j (or h_k) is violated for the ith individual
= 0 otherwise

$$C^2_{con,i} = \frac{\sum_{j=1}^{J}\left[\frac{max\left(g_{j,i}(x),0\right)}{\max_{i=1}^{M}\left[max\left(g_{j,i}(x),0\right)\right]}\right] + \sum_{k=1}^{K}\left[\frac{\left|h_{k,i}(x)\right|}{\max_{i=1}^{M}\left(\left|h_{k,i}(x)\right|\right)}\right]}{J+K} \tag{5}$$

where $g_{j,i}$ is the j^{th} inequality constraint value for the i^{th} individual, and $h_{k,i}$ is the k^{th} equality constraint value for the i^{th} individual. According to Kurapati et al. (2000), Eq.(3)-(5) take both the number of violated constraints $C^1_{con,i}$ and the amount of constraint violation $C^2_{con,i}$ into account in order to measure the quality of individual i with respect to constraint violation in a more accurate way. Also, according to the experimental results from Kurapati et al. (2000), the information regarding the number of violated constraints should be considered to be more important than the amount of constraint violation. As one can see, $C^1_{con,i}$ ranges from 0 to any integer number that is less than the total number of constraints while $C^2_{con,i}$ is scaled ($C^2_{con,i}\leq 1$) by using Eq. (5). If an individual i violates more number of constraints than individual j does, $C^1_{con,i} > C^1_{con,j}$, then $C_{con,i}$ will always be greater than $C_{con,j}$ which implies that individual j is preferred over i. Since $C^2_{con,i}$ is always less than or equal to one, it is only used when two individuals violate the same number of violated constraints.
3) Take new design objectives as: Minimize $f_{new}(x)=\{f_{obj}(x), f_{con}(x)\}$, where $f_{obj}(x_i) = D_{obj,i}$, and $f_{con}(x_i) = C_{con,i}$. For individual i (i =1,…,M), compute the dominant value $D_{new,i}$ in this objective-constraint space.
4) Rank individual i (i =1,…,M) according to its new dominant value $D_{new,i}$. Let $R_i = D_{new.i}$, (i =1,…,M). For those individuals with the same rank R_i, modify their ranks according to their extent of constraint violation $C_{con,i}$. Those individuals with a smaller constraint violation are given higher ranks. In other words, for all the individual s, s+1,…,s+n-1 ($1\leq s\leq M$, $1\leq n\text{-}1\leq M$) with $R_s=R_{s+1}=\ldots=R_{s+n\text{-}1}$, if there exists $C_{con,s}\leq C_{con,s+1}\leq\ldots\leq C_{con,s+n\text{-}1}$, then the rank R'_{s+k} ($0\leq k \leq n\text{-}1$) is:

$$R'_{s+k} = R_s + \frac{k-1}{n} \tag{6}$$

5) Assign a primary fitness value to all individuals according to Eq. (7)

$$P\_fitness_i = C_{max} - (C_{max} - C_{min})\frac{R'_i}{R'_{max}} \tag{7}$$

where: $C_{\max} = 1.2$; $C_{\min} = 0.8$; R'_i is the modified rank of individual i; $R'_{\max}$ is the maximum modified rank when all the individuals in the population are considered, i.e., $R'_{\max}=(R'_1, R'_2,\ldots, R'_M)$.

3.2 SECONDARY FITNESS ASSIGNMENT

The secondary fitness assignment takes place after the first parent is selected from the population using the primary fitness. The concept of "matching" between an individual and the selected first parent is taken into account. Heuristics have been developed, as will be described in this section, to rank individuals based on how much they match with the first parent. The secondary fitness can then be calculated based on individuals' ranks. Those individuals with a high secondary fitness will tend to be selected as the second parent and in turn produce offsprings. The purpose is to generate offsprings that are likely to be better than both parents.

In a MOGA, in order to achieve a fast evolutionary process, it is desired that an offspring be better than both of the parents. Usually, if two parents have some complementary features (e.g., one parent is good from the design objectives point of view while the other from the constraint violation), then it is hoped that their offspring will be better than the two parents since it should inherit the complementary good genes from both parents. (Note:

Of course, there is also a risk of having the offspring being worse than the two parents, since it might inherit the complementary bad genes from both parents. Although throwing the bad children away may appear to encourage fast convergence, our experimental results show that it would indeed be less efficient than simply leaving the bad children in the population, since identifying the bad children also can be computationally expensive.) Here, the two parents with complementary features are regarded as the matching parents. Using the primary fitness alone does not take this matching concept into account. For example, Figure 2 shows a solution set {*A*, *B*, *C*, *D*, *E*, *F*} in an objective-constraint space. According to the procedure described in Section 3.1, the rank order of these six solutions should be $R_F<R_A<R_B<R_C<R_D<R_E$, hence, their primary fitness values should satisfy $P\_fitness_F>P\_fitness_A>\ldots>P\_fitness_E$. If the selection is conducted independently based on the individuals' primary fitness only, when solution *A* is selected as one of the parents, between solutions *E* and *D*, solution *D* will have a higher chance to be selected as the other parent. But, since *A* and *E* are non-inferior to each other, the offspring from *A* and *E* will have a higher chance to receive good complementary features from both parents and in turn a higher chance to be better than both parents, from both objectives and constraints points of view. On the other hand, since *A* dominates *D*, the offspring from *A* and *D* will have a much less chance to perform better than both parents. Hence, from the evolution's point of view, *E* "matches" *A* more than *D* does, although *D* does have a higher primary fitness.

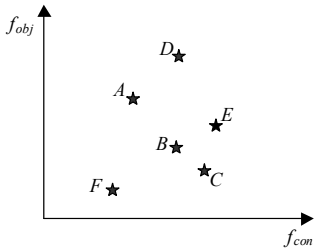


Figure 2: An Example

In order to take the concept of matching into account, the secondary fitness should be used as a complement to the primary fitness. The secondary fitness of an individual expresses the degree of matching between two parents. The individual that matches the first parent most will be assigned the highest secondary fitness value. The following rules are used to rank individuals based on their degree of matching with the first parent.

(i) Individuals that are non-inferior to the first parent in the objective-constraint domain will be given higher ranks because they are considered to match more with the first parent than others do. For instance, as the example shows in Figure 2, if *A* is selected to be the first parent, *B*, *C* and *E* will be considered to match more with *A* than *D* and *F* will, since *B*, *C*, and *E* are all non-inferior with respect to *A*.

(ii) Among all the non-inferior individuals of the first parent, the number of "common" constraints that both a candidate individual and the first parent violate is calculated. The candidate individuals who violate fewer numbers of common constraints are considered to match more with the first parent than others do since they have more complementary features with the first parent, just from the constraints' point of view. As an example, let S_A, S_B and S_C denote the set of constraints that *A*, *B* and *C* violate, respectively. Let $N_{com}(A,B)$ denote the number of common constraints that both *A* and *B* violate. That is: $N_{con}(A,B)=<S_A \cap S_B>$ and $N_{con}(A,C)=<S_A \cap S_C>$, wherein the quantity '$<\bullet>$' refers to the number of elements in the set '$\bullet$'. If there exists $N_{con}(A,B)<N_{con}(A,C)$, then *B* is considered to match more with *A* than *C* does.

(iii) If the numbers of common violated constraints are the same, then those individuals with a less extent of constraint violation will be considered to match more with the first parent. In other words, if there exist: $N_{con}(A,B)=N_{con}(A,C)$, and $C_{con,B}<C_{con,C}$, then *B* is considered to match more with *A* than *C* does.

(iv) Among all the inferior and dominant individuals of the first parent, those with a less extent of constraint violation are considered to match with the first parent more. In the example shown in Figure 2, since there is $C_{con,F}<C_{con,D}$, the individual *F* is considered to match more with the first parent than *D* does.

By using the above rules, individuals can be easily sorted according to the "matching" concept. The secondary rank of the i^{th} individual SR_i is defined as the number of other individuals in the population that are considered to match the first parents less. For instance, if there exist $p_1 \succ p_2 \succ \ldots \succ p_s \succ p_{s+1} \succ \ldots \succ p_M$, wherein "$\succ$" stands for "matches with the first parent more than", then SR_s will have a value of *M-s*. The secondary fitness can then be assigned to all the individuals according to Eq. (8)

$$S\_fitness_i = \frac{SR_i}{SR_{\max}} \tag{8}$$

where $SR_{\max}$ is the maximum secondary rank when all the individuals in the population are considered.

4 A COMPARISON STUDY: MOGA-NCH VS. MOGA-I4

In this section, a baseline MOGA with the new constraint handling technique, hereafter called MOGA-NCH, is compared with the same baseline MOGA where the constraint handling technique CH-I4 (Kurapati et al, 2000), hereafter called MOGA-I4, is implemented. In order to compare the quality of the observed Pareto solution sets from these two algorithms, some new set quality metrics as well as some modified set quality metrics are presented in this section. To make the comparison meaningful, confidence intervals for all metrics in the study are obtained so that the stochastic nature of the MOGA is accounted for.

4.1 SET QUALITY METRICS

In a recent paper by Wu and Azarm (2000), a few set quality metrics were introduced to aid in a quantitative assessment of the quality of an observed Pareto solution set. The metrics used in that paper are: *coverage difference* that evaluates the difference between the size of the objective space dominated by an observed Pareto solution set and that of the objective space dominated by the true Pareto solution set; *Pareto spread* that quantifies how widely the Pareto solution set spreads over the objective space; *accuracy of an observed Pareto frontier* that provides knowledge of an observed Pareto frontier in addition to the observed Pareto solution set; *number of distinct choices* that measures the number of designs that are sufficiently distinct from one another; and *cluster* that expresses the cluster phenomenon. In this section, some of the above metrics are modified so that the quality of the observed Pareto solution set can be assessed more accurately in the presence of multiple observed Pareto solution sets obtained by different optimizers. A new set quality metric, i.e., an inferiority index, is also introduced.

4.1.1 Modified Pareto Spread (*MOS* and MOS_k)

Two metrics in the category of Pareto spread were introduced in Wu and Azarm (2000). These were the *overall Pareto Spread* (*OS*) and the k^{th} *objective Pareto Spread* (OS_k). The overall Pareto Spread quantifies how widely the observed Pareto solution set spreads over the objective space when the design objective functions are considered altogether. The k^{th} objective Pareto spread OS_k aims at quantitatively depicting the Pareto range with respect to each individual design objective (Wu and Azarm, 2000). An observed solution set with higher values of *OS* and OS_k is preferred more than a set with lower values.

When there are multiple observed Pareto solution sets (obtained from different optimizers), some of the observed Pareto solutions might no longer be Pareto and can be inferior with respect to the solutions contained in the remaining Pareto solution sets. Hence, in order to accurately measure the Pareto spread, these new inferior solutions should not be considered. In other words, Pareto spread should be measured over the Meta Pareto solutions instead of the observed Pareto solutions obtained from a single optimizer.

Let P_1, P_2, …, P_T denote all observed Pareto solution sets that need to be compared. The total Meta Pareto solution set is $TMP=\{\ldots,p_{k,j},\ldots,p_{s,l},\ldots\}$ where $p_{j,k}\in P_k$, $p_{l,s}\in P_s$ ($j=1,\ldots,np_k$, $l=1,\ldots,np_s$ and $k,s=1,\ldots,T$). For an observed Pareto solution set P_s, the modified overall Pareto spread (*MOS*) and modified k^{th} objective Pareto spread (MOS_k) are defined as:

$$MOS(P_s)=\frac{\prod_{i=1}^{m}\left|\max_{l=1}^{nm_s}(p_{l,s})_i-\min_{l=1}^{nm_s}(p_{l,s})_i\right|}{\prod_{i=1}^{m}\left|(p_b)_i-(p_g)_i\right|} \tag{9}$$

$$MOS_k(P_s)=\frac{\left|\max_{l=1}^{nm_s}((p_{l,s})_k)-\min_{l=1}^{nm_s}((p_{l,s})_k)\right|}{\left|(p_b)_k-(p_g)_k\right|} \tag{10}$$

where nm_s ($nm_s \le np_s$) stands for the number of Meta Pareto solutions in the observed Pareto set P_s.

For example, in a two-objective space shown in Figure 3, the modified overall Pareto spread and the modified k^{th} objective Pareto spread for the observed Pareto solution set P_1 is calculated as:

$$MOS(P_1)=\frac{h_1h_2}{H_1H_2} \tag{11}$$

$$MOS_1(P_1)=\frac{h_1}{H_1} \qquad MOS_2(P_1)=\frac{h_2}{H_2} \tag{12}$$

where $h_1=\left|f_{1max}^M-f_{1min}^M\right|$, $h_2=\left|f_{2max}^M-f_{2min}^M\right|$, $H_1=\left|(p_g)_1-(p_b)_1\right|$ and $H_2=\left|(p_g)_2-(p_b)_2\right|$.

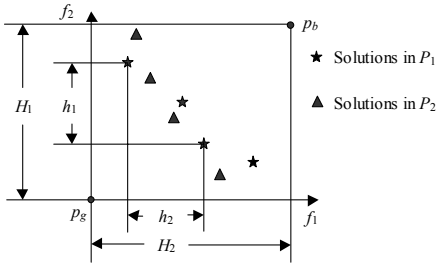


Figure 3: Modified Pareto Spread

4.1.2 Modified Accuracy of the Pareto Frontier (*MAC*)

Knowledge of a Pareto frontier that corresponds to an observed Pareto solution set may become important to a designer dealing with engineering problems, since it might provide some insights into where the potential tradeoff solutions might be. In Wu and Azarm (2000), the observed Pareto solutions were used to approximate the Pareto frontier. With the presence of the other observed Pareto solution sets, Meta Pareto solutions should be used when calculating the quantity for the accuracy of the Pareto frontier.

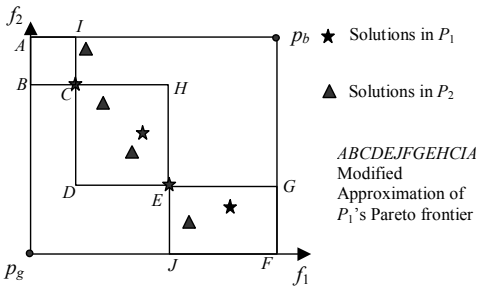


Figure 4: Modified Approximation of the Pareto Frontier

Figure 4 shows the concept of the modified approximation of the Pareto frontier (*MAP*) in a two-

dimensional objective domain. *MAP* can be calculated according to Eq. (13).

$$MAP(P_s) = space(I - S_{in}(P_s^M) - S_{do}(P_s^M)) = space(I) - space(S_{in}(P_s^M)) - space(S_{do}(P_s^M)) \quad (13)$$

wherein I is the entire objective space under consideration, $S_{in}(P_s^M)$ and $S_{do}(P_s^M)$ are the inferior region and dominance region of the Meta Pareto sets P_s^M. The definitions of both inferior region and dominance region of a set can be found in Wu and Azarm (2000). Eqs. (14) and (15) can be used to calculate the space of $S_{in}(P_s^M)$ and $S_{do}(P_s^M)$ respectivly.

$$space(S_{in}(P_s^M)) = \sum_{r=1}^{nm_s}\left\{(-1)^{r+1}\left[\sum_{k_1}^{nm_s-r+1}\cdots\sum_{k_l=k_{l-1}+1}^{nm_s-(r-l+1)+1}\cdots \sum_{k_r=k_{r-1}+1}^{nm_s}\prod_{i=1}^{m}\left[f_i(x_b) - \max_{j=1}^{r}(f_i(x_{k_j}))\right]\right]\right\} \quad (14)$$

$$space(S_{do}(P_s^M)) = \sum_{r=1}^{nm_s}\left\{(-1)^{r+1}\left[\sum_{k_1}^{nm_s-r+1}\cdots\sum_{k_l=k_{l-1}+1}^{nm_s-(r-l+1)+1}\cdots \sum_{k_r=k_{r-1}+1}^{nm_s}\prod_{i=1}^{m}\left[\min_{j=1}^{r}(f_i(x_{k_j})) - f_i(x_g)\right]\right]\right\} \quad (15)$$

where x_g and x_b denote the good point and bad point, nm_s stands for the number of Meta Pareto solutions in observed Pareto set P_s. The quantity *MAC* is taken as the inverse of *MAP*. Hence, the solution set with a higher *MAC* value indicates that an approximation of the Pareto frontier is more accurate and therefore more preferred.

4.1.3 Inferiority Index (*InfI*)

Once the observed Pareto solution set is obtained, the designer will have to choose a preferred solution from the set. The higher the number of true Pareto solutions and the fewer the number of inferior solutions the observed set contains, the higher chance is there for the designer to select a true Pareto solution. With the presence of many Pareto solution sets, although one still can not guarantee that the Meta Pareto solutions are true Pareto solutions, one can be sure that the inferior solutions are not true Pareto solutions. As such, it is desired that a MOGA should provide as fewer inferior solutions as possible and as more Meta Pareto solutions as possible. The metric of inferiority index (*InfI*) is defined as the ratio between the number of inferior solutions that a particular MOGA has obtained over the total number of observed Pareto solutions the MOGA has provided as shown in, Eq. (16).

$$InfI(P_s) = \frac{< P_s > - < P_s^M >}{< P_s >} \quad (16)$$

wherein P_s is the observed Pareto solution set, P_s^M is the Meta Pareto solution set contained in P_s.

A MOGA with a smaller inferiority index implies that it will give the designers a higher chance to select a true Pareto solution from the set.

4.2 CONFIDENCE INTERVALS

Since a MOGA is essentially a stochastic method that has randomness in almost every stage of its evolutionary process, it will be inappropriate to state that one MOGA is better than another solely based on the values of the quality metrics obtained from one single MOGA run. As such, confidence intervals (Longley-Cook, 1985), with a 95% confidence level, are obtained for the set quality metrics based on some random runs of the two MOGAs that are compared. The confidence intervals are scaled such that the mean of MOGA-I4 with respect to each quality metric has a value of 1. The generation of initial population is chosen as the random factor. All other MOGA parameters, such as mutation rate, crossover probability, etc., are kept fixed for both techniques. The values of these fixed parameters are shown in Table 1.

Table 1: MOGA Parameters and Configurations

MOGA Parameters	Configurations
Population size	100
Percentage replacement	10
Crossover type	Two-points crossover
Crossover probability	0.95
Mutation probability	0.05
Selection Type	Tournament selection
Duplication	None

In general, with the confidence intervals for each quality metric, one can clearly see which technique works better without ignoring the random behavior of MOGAs.

5 EXAMPLE

In this section, both MOGA-NCH and MOGA-I4 are tested on an engineering example: design of a vibrating platform. The confidence intervals are obtained for the aforementioned quality metrics to demonstrate the statistical significance of the improvement by MOGA-NCH over MOGA-I4.

5.1 VIBRATING PLATFORM: PROBLEM DESCRIPTION

This example was selected from Messac (1996) with some modifications. It consists of a pinned-pinned sandwich beam with a vibrating motor on its top. As shown in Figure 5, the beam has five layers of three different materials. There is a middle layer and two sandwich layers. The distance from the center of the beam to the outer edge of each layer comprises three of the sizing design variables, d_1, d_2, and d_3. The width of the beam, b, and the length of the beam, L, are the other two sizing design variables. There are also three combinatorial variables for the material type M_i, where i=1,2,3, for the different materials that can be used for each layer. Hence, there are 8 design variables, 3 combinatorial variables for the material type of the 3 layers, and 5 sizing variables.

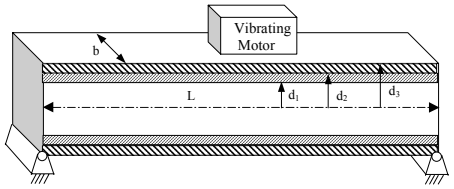


Figure 5: Vibrating Platform Example

The two design objectives are to maximize the fundamental frequency of the beam, and to minimize the material cost. The maximization of the fundamental frequency is converted to a minimization form by minimizing the negative of the fundamental frequency. The problem formulation is shown below:

Minimize $f_1(d_1,d_2,d_3,b,L,M_i) = -(\pi/2L^2)(EI/\mu)^{0.5}$

$$(EI) = (2b/3)[E_1 d_1^3 + E_2(d_2^3 - d_1^3) + E_3(d_3^3 - d_2^3)]$$

$$(\mu) = 2b[\rho_1 d_1 + \rho_2(d_2 - d_1) + \rho_3(d_3 - d_2)]$$

Minimize $f_2(d_1,d_2,d_3,b, M_i) = 2b[c_1 d_1 + c_2(d_2 - d_1) + c_3(d_3 - d_2)]$

Subject to:

$$g_1: \quad \mu L - 2800 \le 0$$
$$g_2: \quad d_2 - d_1 - 0.15 \le 0$$
$$g_3: \quad d_3 - d_2 - 0.01 \le 0 \qquad (17)$$

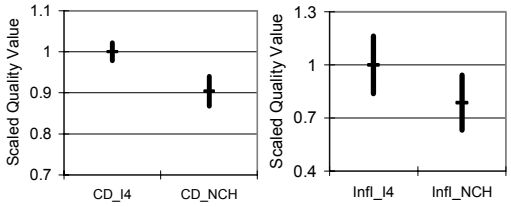
where $0.05 \le d_1 \le 0.5$, $0.2 \le d_2 \le 0.5$, $0.2 \le d_3 \le 0.6$, $0.35 \le b \le 0.5$ and $3 \le L \le 6$. Here, E_i is the modulus of elasticity of material M_i, while ρ_i is the density, and c_i is the cost. According to the material type variable M_i, the value of the parameters E_i, ρ_i, and c_i is different for different layer material, as given in Table 2. It is assumed that the material types for the three layers are mutually exclusive. In other words, the same material cannot be used for more than one layer. However, the layers are allowed to have zero thickness. The first three constraints refer to upper bounds on the mass of the beam, thickness of layer 2, and thickness of layer 3, respectively, and they are labeled g_1 through g_3. The last 5 constraints are the set constraints on the sizing variables (Azarm et al., 1999).

Table 2 Layer's Material Properties of the Vibrating Platform Example

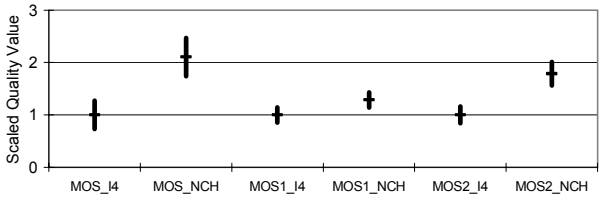
Material M_i	ρ_i (Kg/m^3)	E_i (N/m^2)	C_i (\$/volume)
1	100	1.6×10^9	500
2	2,770	70×10^9	1,500
3	7,780	200×10^9	800

5.2 RESULTS

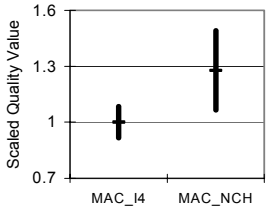
Fifty runs with different randomly generated initial population are performed. The configurations of MOGA parameters are shown in Table 1. Both techniques were run until the number of function evaluations reached 4000. The quality metrics were obtained for all the observed Pareto solution sets. The statistical results are shown in Figure 6. As one can see, MOGA-NCH gives clearly better coverage difference which indicates that the Pareto solution sets from MOGA-NCH are more similar to the true Pareto solution set. MOGA-NCH is more likely to give a smaller inferiority index than MOGA-I4 does which shows that using MOGA-NCH, the designers are less likely to select an inferior solution. The solutions from MOGA-NCH clearly spread over wider space in the objective domain. Moreover, the Pareto frontiers that MOGA-NCH approximates is more accurate than MOGA-I4. Although MOGA-NCH does provide fewer distinct choices than MOGA-I4 does, the solutions from MOGA-NCH are less clustered.



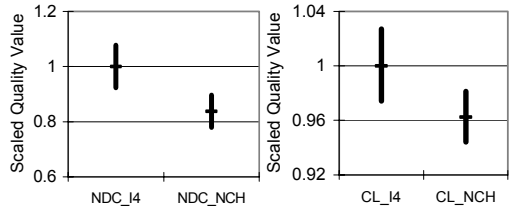
(a) Coverage Difference (*CD*) (b) Inferiority Index (*InfI*)



(c) Modified Overall Pareto Spread (*MOS*) and k^{th} Objective Pareto Spread (MOS_k)



(d) Modified Accuracy of Pareto Frontier (*MAC*)



(e) Number of Distinct Choices (*NDC*) (f) Cluster (*CL*)

Figure 6: Vibrating Platform Results

6 CONCLUSION

In this paper, a new constraint handling (NCH) technique is proposed to work with MOGAs. The technique is based on a primary and secondary assignment schemes. A Pareto ranking scheme is used in the primary fitness assignment so that subjective and problem dependent

parameters are avoided. Rules that take the concept of "matching" into account are then used in a secondary fitness assignment scheme. A MOGA that uses this new constraint handling technique (MOGA-NCH) and a MOGA that uses a previously published constraint handling technique (MOGA-I4) are tested on an engineering example: design of a vibrating platform. The results are compared by using some new set quality metrics. By calculating the confidence intervals, it is observed that for this example, MOGA-NCH performs better than MOGA-I4 with respect to almost all quality metrics, such as coverage difference, inferiority index, Pareto spread, accuracy of the Pareto frontier as well as the cluster, except for the number of the distinct choices.

7 ACKNOWLEDGEMENTS

The work presented in this paper was supported in part by NSF, ONR, IHDIV-NSWC, and Maryland Industrial Partnerships. Such support does not constitute an endorsement by the funding agency of the opinions expressed in the paper.

8 REFERENCES

Azarm, S., Reynolds, B., and Narayanan, S. (1999). Comparison of Two Multi-objective Optimization Techniques with and within Genetic Algorithm. In *Proceedings of the ASME DETC, Design Automation Conference*, Paper No. DETC99/DAC-8584, DETC'99 September 12-16, Las Vegas, Nevada.

Binh, T.T., and Korn, U. (1997). MOBES: A multiobjective evolution strategy for constrained optimization problems. *Third International Conference on Genetic Algorithms (Mendel 97)*, Brno, Czech Republic, pp. 176-182.

Coello, C.A. (1999). A Comprehensive Survey of Evolutionary-Based Multi-objective Optimization Techniques. *Knowledge and Information Systems An International Journal*, Vol. 1, No.3, pp.269-308.

Coello, C.A. (2000). Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, Vol. 32, No.3, pp.275-308.

Fonseca, C.M., and Fleming, P.J. (1993). Genetic Algorithms for Multi-objective Optimization: Formulation, Discussion and Generalization. In Forrest, S., (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, pp. 416-423..

Goldberg, E. D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Co.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, The University of Michigan Press.

Homaifar, A., Qi, C.X., and Lai, S.H. (1994). Constrained Optimization via Genetic Algorithms. *Simulation*, Vol. 62, No.4, pp. 242-254.

Kurapati, A., Azarm, S., and Wu, J. (2000). Constraint Handling in Multi-objective Genetic Algorithms. *Proceedings of 8th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Paper No. AIAA-2000-4893, Long beach, CA, September 6-8.

Longley-Cook, L.H. (1985). *Statistical Problems And How to Solve Them*. Barnes & Noble Books, A Division of Harper & Row, Publishers.

Messac, A. (1996). Physical Programming: Effective Optimization for Computational Design. *AIAA Journal* 34 (1), pp. 149-158.

Michalewicz, Z., and Attia, N. (1994). Evolutionary Optimization of Constrained Problems. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, World Science, pp. 98-108.

Michalewicz, Z. (1995). A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, MIT, pp. 135-155.

Miettinen, K.M. (1999). *Nonlinear Multi-objective Optimization*. Kluwer academic Publishers, Boston.

Narayanan, S., and Azarm, S. (1999). On Improving Multi-objective Genetic Algorithms for Design Optimization. *Structural Optimization*, pp. 146-155.

Schaffer, J.D. (1985). Multiple Objective Optimization With Vector Evaluated Genetic Algorithms. In: *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, Lawrence: Erlbaum, pp. 93-100.

Surry, P.D., Radcliffe, N.J., and Boyd, I.D. (1995). A Multi-Objective Approach to Constrained Optimization of Gas Supply Networks: The COMOGA Method. In Terenee C. Fogarty, editor, *Evolutionary Computing, AISB Workshop*. Selected Papers, Lecture Notes in Computer Science, Springer-Verlag, Sheffild, U.K, pp. 166-180.

Van Veldhuizen, D.A. (1999). *Multi-objective Evolutionary Algorithm: Classifications, Analyses, and New Innovations* Ph.D. Dissertation, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Wu, J., and Azarm, S. (2000), Metrics For Quality Assessment of A Multi-objective Design Optimization Solution Set. In *Proceedings of the ASME DETC, Design Automation Conference*, Paper No. DETC2000/DAC-14233, DETC'2000 September 10-14, Baltimore, Maryland.

Zitzler. E., and Thiele, L. (1998). Multi-objective Optimization Using Evolutionary Algorithms – A Comparative Case Study. In Eiben, A.E., et al., *Proceedings of the 5th International Conference: Parallel Problem Solving from Nature – PPSNV*, Amsterdam, The Netherlands, Springer, pp. 292-301.