Increasing the Robustness of Distributed Genetic Algorithms by Parallel Cooperative-Competitive Genetic Operators

Hernán E. Aguirre **Kiyoshi Tanaka** **Shinjiro Oshita**
Faculty of Engineering, Shinshu University
4-17-1 Wakasato, Nagano, 380-8553 JAPAN
{{ahernan, ktanaka}@gipwc, oshita@oshan}.shinshu-u.ac.jp

Abstract

In this work we study the performance of a distributed GA that incorporates in its core parallel cooperative-competitive genetic operators. A series of controlled experiments are conducted using various large and difficult 0/1 multiple knapsack problems to test the robustness of the distributed GA. Simulation results verify that the proposed distributed GA compared with a canonical distributed GA significantly gains in search speed and convergence reliability with less communication cost for migration.

1 INTRODUCTION

The development of parallel implementations of algorithms has been mainly motivated by the desire to reduce the overall time to completion of a task by distributing the work implied by a given algorithm to processing elements working in parallel (Martin et al., 1997). An alternative approach explores parallel computational models that can exploit interactions among primitive components inducing emergent synergetic behaviors for the entire system (Forrest, 1990).

There are a variety of models for parallelizing Genetic Algorithms (GAs) in the literature. They have been separated in four main categories: global master-slave, island, cellular, and hierarchical parallel GAs (Gordon & Whitley, 1993; Lin et al., 1994; Cantú-Paz, 1998). In a global master-slave GA there is a single population and the evaluation of fitness is distributed among several processors. Selection, crossover and mutation consider the entire population (Hausser & Männer, 1994). A cellular or fine-grained GA consists of one spatially structured population. Selection and mating are restricted to a small neighborhood and the neighborhoods are allowed to overlap permitting some interaction among individuals (Manderick & Spiessens, 1989; Mühlenbein, 1989). An island GA, also known as coarse-grained or distributed GA, consists on several subpopulations evolving separately with occasional migration of individuals between subpopulations (Pettey et al., 1987; Tanese, 1987; Martin et al., 1997). Finally, a hierarchical parallel GA combines an island model with either a master-slave or cellular GA (Cantú-Paz, 1998).

The global master-slave GA does not affect the behavior of the algorithm and can be considered only as a hardware accelerator. However, the other parallel formulations of GAs are very different from canonical GAs (Holland, 1975; Goldberg, 1989), especially with regards to population structure and selection mechanisms. These modifications change the way the GA works affecting its dynamic and the trajectory of evolution. For example, the subpopulation size, migration rate, and migration frequency are crucial to the performance of island models. Cellular, island and hierarchical models perform as well as or better than canonical versions and have the potential of being more than just hardware accelerators (Gordon & Whitley, 1993; Lin et al., 1994; Cantú-Paz, 1998).

Another aspect of GAs that can be parallelized is the application of crossover and mutation. These operators, from a processing time stand, are usually simple and any gain we might expect reducing the overall time to completion could seem minor. However, the processing time viewpoint alone misses the dynamics that can arise from operators with complementary roles acting in parallel. The balance between crossover and mutation is crucial to the performance of GAs. One way to pursue better balances, and therefore better performance, is to combine crossover with higher mutation rates. Higher mutations parallel to crossover can give an efficient framework towards this goal, in which the strengths of the individual operators can be kept without interfering one with the other. Rather than as a hardware accelerator, the more significant gains from the parallel application of operators within parallel GAs could come from exploiting in a better way the interaction between them.

In this work we focus on distributed GAs and study the performance of a distributed GA that incorporates in its core parallel cooperative-competitive genetic operators. We conduct a series of controlled experiments using various large and difficult 0/1 multiple knapsack problems to test the robustness of the distributed GA. Simulation results verify that the proposed distributed GA compared with a canonical distributed GA significantly gains in search speed and convergence reliability with less communication cost for migration.

2 DISTRIBUTED GA (ISLAND MODEL)

The island model GA consists on several subpopulations evolving separately and concurrently with occasional migration of individuals between subpopulations (Martin et al., 1997). Selection, recombination and mutation are applied within each subpopulation. The basic island model uses the same values for crossover and mutation rates in all subpopulations. However, different values for these parameters can be chosen for each subpopulation (Tanese, 1987). Migration of individuals is controlled by several parameters such as: (i) the communication topology that defines the connections between subpopulations, (ii) a migration rate that controls how many individuals migrate, and (iii) a migration interval that affects the frequency of migration. Also, migration must include strategies for migrant selection and for their inclusion in their new subpopulations.

The communication topology can be defined as a graph in which the subpopulations P_i $(i = 0, 1, ..., K-1)$ are the vertices and each defined edge $L_{j,k}$ specifies a communication link between the incident vertices P_j and P_k (neighbor subpopulations). In general, assuming a directed graph, for each defined link $L_{j,k}$ we can indicate the number of individuals $R_{j,k}$ that will migrate from P_j to P_k (migration rate) and the number of generations M between migration events (migration interval). The communication topology and migration rates could be static or dynamic and migration could be asynchronous or synchronous. Various strategies for choosing migrants, such as selection of the best and random selection, have been applied.

The basic island model considers an overall population of λ_{total} individuals that is partitioned into K subpopulations. For an even partition each subpopulation has $\lambda = \lambda_{total}/K$ individuals. It also considers a static topology that is specified at the beginning of the run and synchronous migration occurring every M generations with a constant migration rate R for each defined link $L_{j,k}$.

Distributed GAs are more complex than single population GAs. The subpopulation size, the communication topology (its degree of connectivity), migration rate, and migration frequency are important factors related to the performance of distributed GAs (Cantú-Paz, 1998). There is some experimental evidence that distributed GAs can produce solutions with similar or better quality than single population GAs while reducing the overall time to completion in an factor that is almost in reciprocal proportion to the number of processors.

3 IMPROVED DGA

3.1 CONCEPT OF GA-SRM

An empirical model of GA that puts parallel genetic operators in a cooperative-competitive stand with each other, pursuing better balances for crossover, mutation, and selection, has proved to be an effective approach to improve the search performance of single population GAs (Aguirre et al., 1999; Aguirre et al., 2000).

The main features of the model are (i) two genetic operators with complementary roles applied in parallel to create offspring: Self-Reproduction with Mutation (SRM), and Crossover and Mutation (CM) (ii) an extinctive selection mechanism, and (iii) an adaptive mutation schedule that varies SRM's mutation rates from high to low values based on SRM's own contribution to the population.

To achieve better balances for mutation and crossover the algorithm should be able to combine crossover with higher mutation rates during the course of a run. One way to try this is to simply apply one operator after the other, as in a canonical GA (Holland, 1975; Goldberg, 1989). Under this configuration, however, there could be interferences between crossover and high mutation. If mutation probabilities are high then although crossover alone could be doing a good job it is likely that some of the just created favorable recombinations may be lost because of the high disruption introduced by mutation. On the other hand, mutation could be working well but crossover may produce poor performing individuals (Spears, 1998) affecting the survivability of beneficial mutations that would contribute to the search.

Another way to combine crossover with higher mutation rates is to apply high mutation parallel to crossover. The parallel formulation of operators can avoid the interferences between operators, implicitly increasing the levels of cooperation to introduce and propagate beneficial mutations. It also sets the stage for competition between operators' offspring. Thus, to create offspring, in the model CM applies crossover followed by background mutation and SRM applies parallel adaptive mutations.

Although the parallel formulation of genetic operators can avoid interferences between operators, it does not prevent SRM from creating deleterious mutations or CM from producing ineffective crossing over operations. To cope with these cases the model also incorporates the concept of extinctive selection that has been widely used in Evolutionary Strategies (Bäck, 1996). Through extinctive selection the offspring created by CM and SRM coexist competing for survival (the poor performing individuals created by both operators are eliminated) and reproduction. The parallel formulation of genetic operators tied to extinctive selection creates a cooperative-competitive environment for the offspring created by CM and SRM.

3.2 GA-SRM INTO DGA

3.2.1 Communication Topology

In this work we use a $+1+...+L$ communication topology (Cantú-Paz, 1999a) in which each subpopulation P_i $(i = 0, 1, ..., K-1)$ is linked to the next L subpopulations. The neighbor populations are defined by the directed links $L_{j,k}$ where

$$k = \{j+1, ..., j+L\} \bmod K. \tag{1}$$

Fig. 1 illustrates a $+1+2$ island model in which each subpopulation is linked to two neighbors ($L = 2$). In this example, for instance, subpopulation P_0 can only send individuals to P_1 and P_2 and receive migrants from P_4 and P_5.

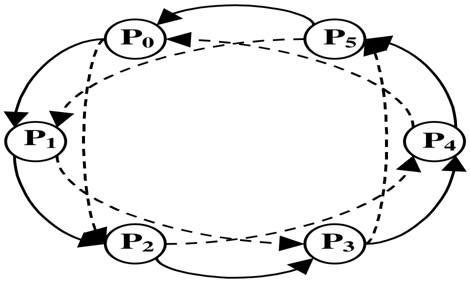


Fig. 1 $+1+2$ communication topology.

3.2.2 CM and SRM in DGA

In the above setting, the inclusion of GA-SRM into a distributed GA is straightforward and the basic form of GA-SRM is mostly preserved in each subpopulation $P_i(t)$ $(i = 0, 1, ..., K-1)$ at the $t-$th generation. CM creates offspring by conventional one-point crossover and successive background mutation operator (Holland, 1975; Goldberg, 1989) in each $P_i(t)$. The same crossover rate p_c is used in all $P_i(t)$. The mutation probability $p_m^{(CM)}$ is set to a constant small value and is also the same in all $P_i(t)$. CM creates λ_{CM} offspring within $P_i(t)$ and is expected to propagate beneficial genetic information into the subpopulation by combining segments from parent individuals.

On the other hand, SRM creates offspring by an adaptive mutation operators called Adaptive Dynamic Segment (ADS) (Aguirre et al., 1999; Aguirre et al., 2000), which directs mutation only to a segment of the chromosome using constant high mutation probabilities per bit in all $P_i(t)$.

$$p_m^{(SRM)} = \begin{cases} \alpha \ (\textit{if the bit is in the segment}) \\ 0 \ (\textit{otherwise}) \end{cases}$$

However, the mutation segment size ℓ_i $(i = 0, 1, ..., K-1)$ is independently adjusted in each $P_i(t)$ based on a normalized mutants survival ratio specified by

$$\gamma_i = \frac{\mu_{SRMi}}{\lambda_{SRM}} \cdot \frac{\lambda}{\mu}, \tag{2}$$

where μ_{SRMi} is the number of SRM's offspring that survive extinctive selection, λ_{SRM} is the offspring number created by SRM, λ is the total offspring number ($\lambda_{CM} + \lambda_{SRM}$), and μ is the number of parent individuals in $P_i(t)$.

In each $P_i(t)$, ℓ_i varies dynamically from n_0 (initial segment mutation size) to $1/\alpha$ by reducing it to ℓ_i/β ($\beta > 1$) every time γ_i falls under a predetermined threshold τ ($\gamma_i < \tau$). Hence, the expected average number of flipped bits goes down from $n_0\alpha$ to 1. Also, the segment initial position, for each chromosome, is chosen at random. SRM is expected to introduce diversity into each subpopulation and its adaptation mechanism to provide better balances for mutation and crossover throughout the course of a run.

3.2.3 (μ, λ) Proportional Selection in DGA

In order to implement the extinctive selection mechanism, (μ, λ) Proportional Selection (Bäck, 1996) is used similar to (Aguirre et al., 1999; Aguirre et al., 2000). Selection probabilities within each subpopulation $P_i(t)$ are computed by

$$Prob(\boldsymbol{x}_j^{(t)}) = \begin{cases} \dfrac{f(\boldsymbol{x}_j^{(t)})}{\sum_{k=1}^{\mu} f(\boldsymbol{x}_k^{(t)})} & (1 \le j \le \mu) \\ 0 & (\mu < j \le \lambda) \end{cases} \tag{3}$$

where $\boldsymbol{x}_j^{(t)}$ is an individual at generation t which has the j-th highest fitness value $f(\boldsymbol{x}_j^{(t)})$ in $P_i(t)$.

3.2.4 Migration Policy

Migration implements a synchronous elitist broadcast strategy (Cantú-Paz, 1999b) occurring every M generations. Each subpopulation broadcasts a copy of its R best individuals to all its neighbor subpopulations. Hence, every subpopulation in every migration event receives $\lambda_m = L \times R$ migrants. In the target subpopulations, the arriving λ_m migrants replace the same number of worst performing individuals and the replacement occurs before extinctive selection. Thus, λ_m migrants also compete to survive with the best $\lambda - \lambda_m$ offspring produced by SRM and CM inside $P_i(t)$. In the following the migration rate is calculated

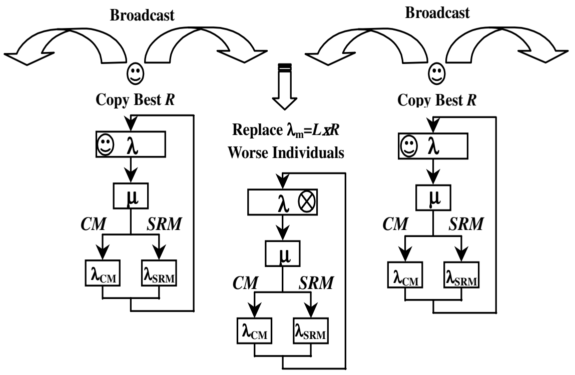


Fig. 2 Migration policy and extinctive selection.

as $100 \times \lambda_m/\lambda$. **Fig. 2** illustrates the migration process to a given subpopulation from its two neighbors (assuming a $+1+2$ communication topology). As mentioned in **3.2.2**, SRM's adaptation occurs locally in each subpopulation $P_i(t)$ but it is not realized at the generations in which migration is performed.

4 0/1 MULTIPLE KNAPSACK PROBLEM

In order to study the robustness of the distributed genetic algorithm we use various large and difficult 0/1 multiple knapsack problems proposed in (Chu & Beasley, 1998) and obtained from the OR-Library (Beasley, 1996).

The 0/1 multiple knapsack problem consists of m knapsacks of capacities $c_1, c_2, ..., c_m$ and n objects. Each object has a profit p_i $(1 \leq i \leq n)$, weights w_{ij} $(1 \leq j \leq m)$, and it is either placed in all m knapsacks or in none at all. The 0/1 multiple knapsack problem can be formulated to maximize the function

$$g(\boldsymbol{x}) = \sum_{i=1}^{n} p_i x_i \tag{4}$$

subject to

$$\sum_{i=1}^{n} w_{ij} x_i \leq c_j \qquad (j = 1, ..., m) \tag{5}$$

where $x_i \in$ {0,1} $(i = 1, ..., n)$ are elements of a solution vector $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, which is the combination of objects we are interested in finding. Solutions to this problem have a natural binary representation in the GA constructed by mapping each object to a locus within the binary chromosome. A 1 in locus i indicates that the object i is being included in the knapsacks and a 0 otherwise. A solution vector $\boldsymbol{x}$ should guarantee that no knapsack is overfilled and the best solution should yield the maximum profit. An $\boldsymbol{x}$ that overfills at least one of the knapsacks is considered as an infeasible solution. The 0/1 multiple knapsack problem is regarded as NP hard combinatorial optimization problem.

Table 1 Genetic algorithms parameters

Parameter	*DGA*	*DGA-SRM*
Selection	Proport.	(μ, λ) Proport.
Scaling	Linear	Linear
Mating	$(\boldsymbol{x}_i, \boldsymbol{x}_j),\ i \neq j$	$(\boldsymbol{x}_i, \boldsymbol{x}_j),\ i \neq j$
p_c	0.6	1.0
$p_m^{(CM)}$	$1/n$	$1/n$
$p_m^{(SRM)}$	-	$\alpha = 0.5,\ \ell = [n/4, 2]$
β	-	2
$\mu : \lambda$	-	$1 : 2$
$\lambda_{CM} : \lambda_{SRM}$	-	$1 : 1$

5 RESULTS AND DISCUSSION

5.1 EXPERIMENTAL SETUP

We test two kinds of distributed GAs in our simulations. (i) a distributed canonical GA (denoted as DGA), and (ii) the proposed distributed GA-SRM (denoted as DGA-SRM). **Table 1** details the parameters used within each subpopulation by DGA and DGA-SRM. DGA implements the same $+1+...+L$ communication topology and migration policy used by DGA-SRM described in **3.2.1** and **3.2.4**.

The knapsack problems we conduct experiments with are highly constrained (Chu & Beasley, 1998). To deal with infeasible solutions a penalty term is introduced into the fitness function. We try the two following fitness functions

$$f_1(\boldsymbol{x}) = g(\boldsymbol{x}) - s \cdot max\{p_i\} \tag{6}$$

$$f_2(\boldsymbol{x}) = \begin{cases} g(\boldsymbol{x})/s \cdot max\{o_j\} & (s > 0) \\ g(\boldsymbol{x}) & (s = 0) \end{cases} \tag{7}$$

where s $(0 \leq s \leq m)$ is the number of overfilled knapsacks and o_j (> 1) is the overfilling ratio of knapsack j calculated by

$$o_j = \sum_{i=1}^{n} w_{ij} x_i / c_j. \tag{8}$$

The fitness function f_1 of **Eq. (6)** did not produce feasible solutions or the results were very poor especially on problems with restrictive knapsack capacity. Similar behavior was observed in (Michalewicz, 1996) for single $(m = 1)$ 0/1 knapsack problems of restrictive capacity. On the other hand, the fitness function f_2 of **Eq. (7)** did produce feasible solutions on all test problems. The results reported here are obtained using f_2 with random initial populations initialized with a 0.25 probability for 1s.

As a point of reference for the quality of solutions, **Table 2** shows results for some of the test problems obtained by the single population versions of the distributed GAs, denoted as cGA and GA-SRM, respectively, set with a population size of $\lambda = 100$ individuals, $T = 10^6$ function evaluations, and SRM's adaptation threshold $\tau = 0.64$. In **Table 2** column *Problem* identifies the problem instance. *Name*

Table 2 Results by single population cGA and GA-SRM

Problem				*cGA*			*GA-SRM*		
$Name$	m	n	ϕ	$Average$	$Stdev$	Gap	$Average$	$Stdev$	Gap
$5.100-00$	5	100	0.25	23271.3	125.75	5.35	24242.8	33.78	1.40
$10.100-00$	10	100	0.25	21846.3	198.94	6.96	22855.4	67.3	2.66
$30.100-00$	30	100	0.25	20494.2	180.49	9.23	21711.8	116.76	3.84
$30.100-10$	30	100	0.50	38509.1	186.49	6.70	40241.9	144.31	2.51
$30.100-20$	30	100	0.75	55348.4	244.34	4.55	57046.1	241.98	1.62
$30.250-00$	30	250	0.25	51920.6	180.11	9.59	55703.9	117.55	3.01
$30.500-00$	30	500	0.25	106023.5	395.15	9.09	113135.3	280.52	2.99

is the name of the problem, m the number of knapsacks, n the number of objects, and ϕ the tightness ratio between knapsack capacities and object weights (restrictiveness of the capacities). *Average* is the average of the best solutions in 10 runs, *Stdev* is the standard deviation around *Average*, and *Gap* indicates the percentage gap between *Average* and the optimal value given by the linear programming relaxation (Chu & Beasley, 1998) (the optimal integer 0/1 solutions for the test problems are not known).

In our study we observe the influence of the problem difficulty, the subpopulation size, and the migration rate on the robustness of the distributed GAs. The distributed GAs use a $\lambda_{total} = 800$ individuals and the same $T = 10^6$ function evaluations. Also, unless indicated otherwise, the distributed GAs use a configuration of $K = 16$ subpopulations ($\lambda = 50$), SRM's adaptation threshold $\tau = 0.56$, and a 10% migration rate.

5.2 PROBLEM DIFFICULTY

Factors related to the difficulty of the problem are the tightness ratio ϕ, the number of objects n, and the number of knapsacks m.

First we observe the performance of the distributed GAs on problems with different ratio ϕ. **Fig. 3** illustrates results by DGA and DGA-SRM on problems of $m = 30$ capacities, $n = 100$ objects, and ratio $\phi = \{0.25, 0.50, 0.75\}$. To present a broader picture this and the subsequent figures plot the error Gap for migration intervals of $M = \{2, 5, 10, 20, 40, 100\}$ generations as well as results when no migration is used and the subpopulations evolve in total isolation (indicated by NM). Results obtained by the single population GAs are also indicated on the left Y axis.

The main conclusions drawn from **Fig. 3** are as follows. (i) The quality of the solutions found by both DGA and DGA-SRM decreases (larger Gap values) as the ratio ϕ is reduced. These results are quite intuitive since reductions on ϕ imply reductions on the fraction of possible subsets of objects that constitute feasible solutions. Consequently, the ratio between the feasible part of the search space and the whole search space gets smaller and the smaller this ratio is the harder it is to find feasible results. This was also observed for 0/1 single ($m = 1$) knapsack problems in (Michalewicz, 1996). (ii) The performance of DGA-SRM without migration is by far superior to DGA without migration indicating that a better search is being carried out within each subpopulation in DGA-SRM. (iii) DGA is far away from DGA-SRM unless DGA uses very short migration intervals (2 generations). DGA-SRM achieves high performance with less communication cost for migration, which could be a big advantage for implementation (note that DGA-SRM even without migration performs better than DGA with migration intervals of 5 generations). It seems that DGA-SRM has an optimum range of migration interval (around $10 \sim 20$, for a 10% migration rate) that attains the minimum error gap. Different from DGA, very short migration intervals (less than 10 generation) deteriorates the performance of DGA-SRM. (iv) DGA and DGA-SRM can achieve better results than its single population versions if migration is included.

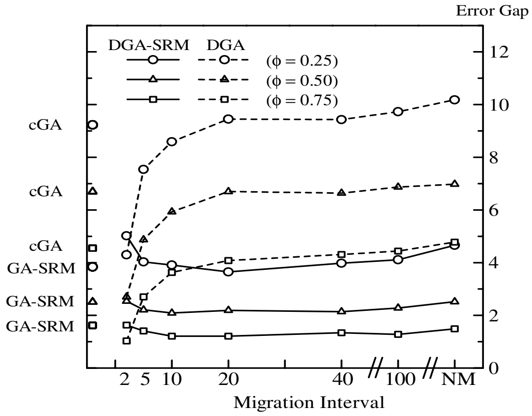


Fig. 3 Tightness of the Capacities ϕ
($K = 16$, 10% migration, $m = 30$, $n = 100$).

Second we fix the number of capacities m and tightness ratio ϕ and observe the effect of the number of objects n. **Fig. 4** illustrates results by DGA and DGA-SRM on problems of $m = 30$ capacities, $n = \{100, 250, 500\}$ objects, and ratio $\phi = 0.25$. From **Fig. 4** we can see that increasing the number of objects n also makes it harder for the algorithms to find high quality solutions. Also, we can clearly see the DGA-SRM's optimum migration interval in this figure. The general behavior by DGA-SRM and DGA are similar to that observed in **Fig. 3**.

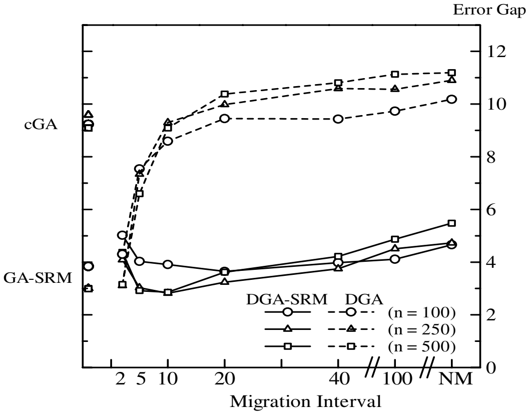


Fig. 4 Objects n
($K = 16$, 10% migration, $m = 30$, $\phi = 0.25$).

Third, we observe the effect of the number of knapsacks m fixing the number of objects n and tightness ratio ϕ. **Fig. 5** illustrates results by DGA and DGA-SRM on problems of $m = \{5, 10, 30\}$ capacities, $n = 100$ objects, and ratio $\phi = 0.25$. Similar to ϕ and n, from **Fig. 5** we can see that increasing the number of knapsacks m has an strong impact on the performance of the algorithms and that between DGA and DGA-SRM the latter exhibits higher robustness. Also, looking at **Fig. 4** and **Fig. 5**, judging from the relative increase on the Gap values, increasing the number of knapsacks (constraints) has an stronger impact than increasing the number of objects (search space).

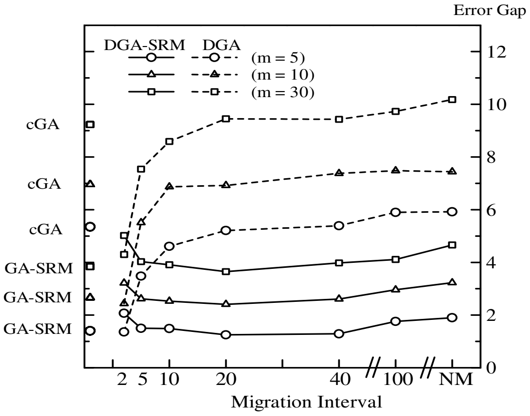


Fig. 5 Knapsacks m
($K = 16$, 10% migration, $n = 100$, $\phi = 0.25$).

5.3 SUBPOPULATION SIZE

Fourth, we choose one problem ($m = 30$, $n = 100$, and $\phi = 0.25$) and observe the effect of reducing the subpopulation size while increasing the number of subpopulations (the overall number of offspring and the total number of function evaluations are kept constant to $\lambda_{total} = 800$ individuals and $T = 10^6$ evaluations). **Fig. 6** illustrates results by DGA and DGA-SRM using $K = \{8, 16, 32\}$ subpopulations with subpopulations sizes of $\lambda = \{100, 50, 25^1\}$ and $\tau = \{0.64, 0.56, 0.50\}$, respectively.

DGA-SRM tolerates population reductions better than DGA (see NM for both algorithms) and can still approach GA-SRM's performance relaying on migration. We could not recognize big performance differences between $K = 8$ and $K = 16$ while smaller subpopulations ($K = 32$) tend to require shorter migration intervals.

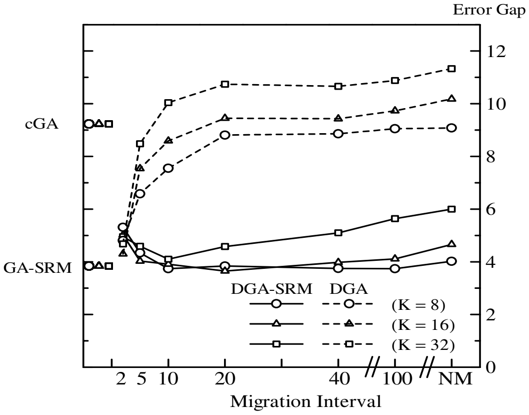


Fig. 6 Subpopulation size λ, K
(10% migration, $m = 30$, $n = 100$, $\phi = 0.25$).

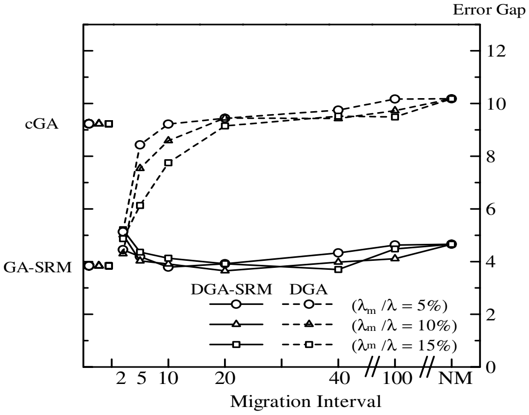


Fig. 7 Migration Rate λ_m/λ
($K = 16$, $m = 30$, $n = 100$, $\phi = 0.25$).

5.4 MIGRATION RATE

Fifth, the effect of the migration rate is also observed. **Fig. 7** illustrates results by DGA and DGA-SRM on one of the problems using migration rates of {5%,10%,15%}.

In DGA-SRM smaller migration rates need shorter migration intervals and vice versa. To reduce communication

---

[1] When $K = 32$ DGA-SRM uses only $\lambda = 24$ to keep a $1:1$ balance for offspring creation between CM and SRM.

cost, it may be better to use larger migration intervals with higher migration rates in DGA-SRM.

5.5 ADAPTATION

Fig. 8 illustrates the adaptation of mutations rates in DGA-SRM. The figure shows the average number of the actual bits flipped by SRM over the generations for some of the subpopulations. From **Fig. 8** we can see that adaptation of mutation rates follow similar trajectories and that for most of the run the mutation rates are higher than the usual expected 1 flipped bit of canonical algorithms. It should also be noticed that the instantaneous averages differ, which is a consequence of the local adaptation within each subpopulation. The local adaptation, besides varying mutation rates, also induces different mutation rates for each subpopulation $P_i(t)$.

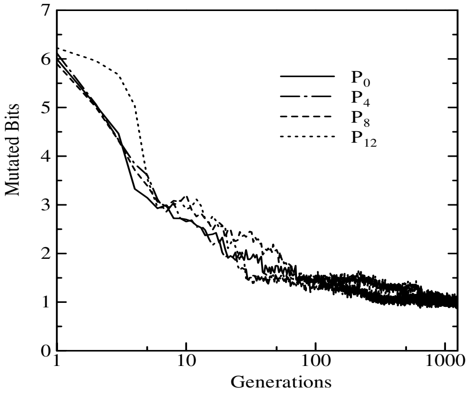


Fig. 8 SRM's adaptation in DGA-SRM
($K = 16, m = 30, n = 100, \phi = 0.25$).

5.6 EXTINCTIVE SELECTION

The remarkable increase in solution quality by the canonical DGA when very short migration intervals are used, i.e. 2 generations, seems at first glance rather counterintuitive (with such migrations intervals one might expect faster convergence but not higher solution quality). However, this is explained by the nature of the test problems and the additional selection intensity caused by migration.

As mentioned above, the problems used in this study are highly constrained with sparse feasible regions where algorithms with penalty functions have a hard time finding feasible solutions (Michalewicz, 1996; Chu & Beasley, 1998). A higher selection pressure in these problems is helping the algorithms to focus the search around the feasible regions (this point has been previously verified in (Aguirre et al., 2000) for single population GAs).

The strategies chosen in this work for migrants selection and replacement (selection of the best and replacement of the worst) causes an increase in the overall selection intensity (Cantú-Paz, 1999b). These strategies combined with very short migration intervals are capable of producing significant selection pressures, which are being used by the DGA. In the case of DGA-SRM, the higher selection pressure is incorporated within the selection mechanism.

Fig. 9 illustrates the effect of extinctive selection in the distributed algorithms and clarifies the contributions of extinctive selection and adaptive mutation SRM in DGA-SRM. We show results by the canonical DGA with $\lambda = 50$ individuals (DGA(50)) in each subpopulation, a DGA using (μ, λ) Proportional Selection with $\mu = 25$ parents and $\lambda = 50$ offspring in each subpopulation (DGA(25,50)), and the DGA-SRM with similar population sizes (DGA-SRM(25,50)). From this figure we see that extinctive selection alone increases the reliability of the distributed GA in this kind of problems. However, when adaptive parallel mutation, SRM, is used the robustness of the algorithm is improved further.

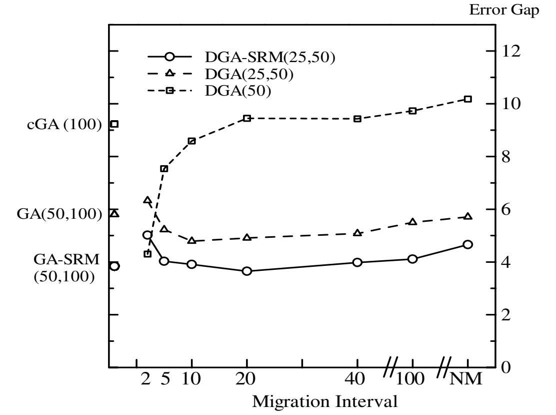


Fig. 9 Effect of Extinctive Selection
($K = 16, m = 30, n = 100, \phi = 0.25$).

Fig. 10 plots the average fitness in the 10 runs of the best solution over the generations by DGA-SRM and DGA. From this figure it can be observed that DGA-SRM has not only higher convergence reliability due to SRM but also a higher search speed caused by extinctive selection.

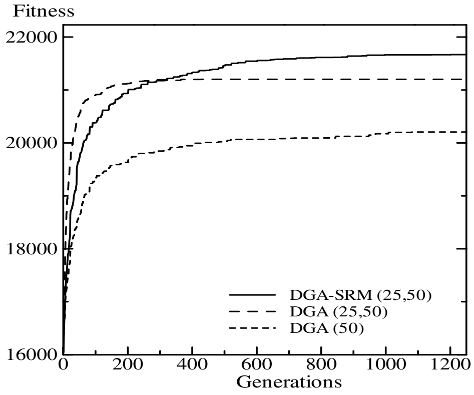


Fig. 10 Fitness Transition
($K = 16, m = 30, n = 100, \phi = 0.25, M = 20$).

6 CONCLUSIONS

In this work we have studied the performance of a distributed GA that incorporates parallel cooperative-competitive genetic operators (DGA-SRM). A series of controlled experiments using various large and difficult 0/1 multiple knapsack problems were conducted to test the robustness of DGA-SRM. Comparisons were made between DGA-SRM and a canonical distributed GA (DGA).

We observed that high selection intensity helps to perform a better search in this kind of combinatorial problems. The DGA-SRM incorporates a higher selection pressure within its selection mechanism. The canonical DGA, however, has to rely in the higher selection intensity introduced by migration and can achieve high results only at the expense of very high communication cost.

The inclusion of high mutation parallel to crossover within DGA-SRM improves further the convergence reliability of the canonical DGA regardless of the difficulty of the problem. Also, due to the high selection intensity within each subpopulation and its built-in source of diversity by SRM, the search speed of the algorithm is increased without sacrificing the quality of solutions. DGA-SRM even without migration produces very high results compared to canonical DGA with small migration intervals.

As future works, the effect that other communication topologies and migration policies have on the performance of this kind of hierarchical GA should be investigated. Also, we would like to extend the concept of GA-SRM to cellular models.

References

H. Aguirre, K. Tanaka and T. Sugimura (1999), “Cooperative Model for Genetic Operators to Improve GAs”, *Proc. IEEE Int’l. Conf. on Information, Intelligence and Systems*, pp.98-106.

H. Aguirre, K. Tanaka, T. Sugimura and S. Oshita (2000), “Cooperative-Competitive Model for Genetic Operators: Contributions of Extinctive Selection and Parallel Genetic Operators”, *Proc. Late Breaking Papers GECCO’2000*, pp.6-14.

T. Bäck (1996), *Evolutionary Algorithms in Theory and Practice*, Oxford Univ. Press.

J. E. Beasley (1996), Obtaining Test Problems via Internet, *Journal of Global Optimization*, vol. 8, pp.429-433.

E. Cantú-Paz (1998), “A Survey of Parallel Genetic Algorithms”, *Calculateurs Paralleles, Reseaux et Systems Repartis*, Vol.10, No.2, pp.141-171.

E. Cantú-Paz (1999a), “Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms”, *Proc. GECCO’99*, pp.91-98.

E. Cantú-Paz (1999b), “Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms”, *Proc. Late Breaking Papers GECCO’99*, pp.65-73.

P. C. Chu and J. E. Beasley (1998), “A Genetic Algorithm for the Multidimensional Knapsack Problem”, *Journal of Heuristics*, vol.4, pp.63-86.

S. Forrest (1990), “Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks”, *Emergent Computation*, *Physica D42*, pp.1-11.

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

V. S. Gordon and D. Whitley (1993), “Serial and Parallel Genetic Algorithms as Function Optimizers”, *Proc. 5th ICGA*, Morgan Kaufmann, pp.177-183.

R. Hausser and R. Männer (1994), “Implementation of Standard Genetic Algorithm on MIMD Machines”, *Parallel Problem Solving from Nature III*, Springer-Verlag, pp.504-513.

J. H. Holland (1975), *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press.

S. C. Lin, W. Punch and E. Goodman (1994), “Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach”, *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*, pp.28-37.

B. Manderick and P. Spiessens (1989), “Fine-Grained Parallel Genetic Algorithms”, *Proc. 3th ICGA*, Morgan Kaufmann, pp.428-433.

W. N. Martin, J. Lienig, and J. P. Cohoon (1997), “Island (migration) models: evolutionary algorithms based on punctuated equilibria”, *Handbook of Evolutionary Computation*, IOP and Oxford University Press, pp.C.6.3:1-16.

Z. Michalewicz (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, third revised and extended edition.

H. Mühlenbein (1989), “Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization”, *Proc. 3th ICGA*, Morgan Kaufmann, pp.416-421.

C. B. Pettey, M. R. Leuze and J. J. Grefenstette (1987), “A Parallel Genetic Algorithm”, *Proc. 2nd ICGA*, Lawrence Erlbaum Assoc., pp.155-161.

W. Spears (1998), *The Role of Mutation and Recombination in Evolutionary Algorithms*, Ph.D. thesis, George Mason University.

R. Tanese (1987), “Parallel Genetic Algorithms for a Hypercube”, *Proc. 2nd ICGA*, Lawrence Erlbaum Assoc., pp.177-183.

An interactive genetic algorithm with co-evolution of weights for multiobjective problems

Helio J.C. Barbosa and **André M.S. Barreto**
Laboratório Nacional de Computação Científica
Av. Getúlio Vargas 333
25651 070 Petrópolis RJ, BRAZIL
`{hcbm,andremsb}@lncc.br`

Abstract

An interactive co-evolutionary genetic algorithm is proposed for a class of multi-objective optimization problems and is applied to the problem of graph lay-out generation taking into account the personal user's preferences.

1 INTRODUCTION

When using evolutionary algorithms (EAs) for solving a given problem, the user has to define a fitness function which will be used to indicate the quality of a given candidate solution. This requirement can be relaxed if the user somehow is able to rank the whole population of candidate solutions according to its quality. As a minimum requirement, given two candidate solutions, the user must be able to tell which one is the best.

In a number of applications for which EAs seem to be a suitable tool, it is very difficult to construct a fitness function which accurately reflects the user's perception of what is a good solution for the problem at hand. For instance, all artistic applications of EAs fall into this category where the fitness function is subjective.

The potential of *interactive* evolution was first demonstrated by Dawkins[1] and latter developed by Sims[2, 3]. Evolution strategies with subjective selection have been developed by Herdy[4, 5]. Additionally, a number of examples can be found in [6]. For musical applications, one is referred to [7, 8, 9, 10, 11] and references therein. Some hard engineering problems have also been approached in such a way that the user is included in the loop of the evolutionary process. For instance, Gruau and Quatramaram[12] have evolved neural nets for robot control by letting the user subjectively affect the fitness value of a given solution.

It is well known to the interactive evolutionary computation (IEC) practitioners that user fatigue is a key element to the success of an IEC application and, as a result, the construction of tools which are able to learn the fitness function/user's preferences and then replace the human in the loop is an attractive idea. Using data gathered during interactive run(s) with the user, neural networks have been trained with the objective of replacing the user and allowing for longer runs[9, 11].

Yet another very important situation where user preferences come into play is the arena of multi-objective optimization (MOO) problems, where genetic algorithms (GAs) are specially attractive and a large literature is available (see the site maintained by Coello: www.lania.mx/~ccoello/EMOO/EMOObib.html). Although GAs have been successfully applied to MOO problems, specially by using its population of solutions to approximate the whole Pareto front, the task of both the GA (in finding the Pareto front) and the user (in choosing a good compromise solution) grows in complexity as the number of objectives grows. One alternative solution is to weight the several objective functions in such a proportion that the resulting scalar objective function leads the GA to eventually find a solution which satisfies the user's preferences. Of course, the task of assigning such weights is not trivial and we believe that the idea of co-evolution can help the design of an interactive GA (IGA) for this class of problem.

One particular MOO application will be considered here, namely that of graph lay-out, in order to test the proposed procedure. An interactive co-evolutionary GA is proposed which maintains two populations: a graph lay-out population and a weight-set population. Each population evolves via an independent GA but their evolution is coupled by their fitness evaluation, which involves active user intervention. For each lay-out several aesthetical objectives are mathematically defined and the final fitness value is obtained by the current set of weights. However, the population of weight-sets also evolves according to a fitness defined as how well a given set of weights ranks the population of lay-outs as compared to the ranking periodically provided by the user. It is expected that the lay-out population improves with respect to the current weight-set and also that the weight-set population evolves generating increasingly better weight-sets, i.e., sets that better reflect the user's preferences.

The paper is organized as follows: Section 2 describes the multi-objective optimization problem and the graph lay-out problem, Section 3 briefly reviews interactive evolutionary computations and Section 4 summarizes the idea of co-evolution and some previous work. Section 5 describes the co-evolutionary GA used here and Section 6 presents some computational experiments. The paper ends with conclusions and suggestions for further work.

2 MULTI-OBJECTIVE OPTIMIZATION

In an unconstrained multi-objective optimization (MOO) problem one seeks to optimize the m components of a vector of objective functions $F(x) = (f_1(x), f_2(x), \ldots, f_m(x))$ where $x \in \Omega \subset R^n$ is the vector of design (or decision) variables which belong to the admissible set Ω. The function $F : \Omega \mapsto \Lambda$ maps solution (or design) vectors $x = (x_1, x_2, \ldots, x_n)$ to vectors $y = (y_1, y_2, \ldots, y_m)$, with $y_i = f_i(x)$, in the objective function space. Without loss of generality it is assumed that each objective is to be minimized. Due to their non-commensurable and conflicting nature there is usually not a solution which minimizes all m objectives simultaneously. This motivates the concept of *dominance*: a vector $a = (a_1, a_2, \ldots, a_n)$ dominates a vector $b = (b_1, b_2, \ldots, b_n)$ if $a_i \leq b_i \, \forall i$ and there is j such that $a_j < b_j$. One then defines Pareto optimality: a solution $x \in D$ is said to be Pareto-optimal in D if and only if there is no $x' \in D$ such that its image $F(x')$ dominates $F(x)$. The Pareto-optimal set, $\mathcal{P}$, is thus the set of all $x \in D$ such that $F(x)$ is non-dominated in Λ. The Pareto front $\mathcal{P}_F$ is the image of the Pareto-optimal set in the objective function space.

The first application of GAs to MOO problems dates back to the mid-eighties[13, 14] and several papers have been published since then (see [15, 16, 17, 18]). An example of a *subjective* MOO problem is presented by Shibuya et all[19] where animations of human-like motions are to be computer generated. Another example is that of graph layout, which is discussed in the following section.

2.1 The graph lay-out problem

The graph lay-out or graph drawing (GD) problem considered here is the task of producing aesthetically-pleasing two-dimensional pictures of undirected graphs drawn with straight edges. Vertices will be drawn as points in the plane inside a rectangular frame and the problem thus reduces to finding the coordinates of such points, since the edges are defined by a given 0/1 connection matrix.

Automatic graph lay-out is a long-studied problem in computer science with a large literature (see [20], for a bibliography). Many aesthetic criteria can be conceived of and some generally accepted ones include: (i) uniform spatial distribution of the vertices, (ii) minimum number of edge-crossings, (iii) uniform edge length, (iv) exhibit any existing symmetric feature and (v) vertices should not be placed too close to edges. Of course one would strive for an algorithm which generates a solution fulfilling all the aesthetic criteria simultaneously. However, those criteria are non-commensurable and conflicting. As an alternative to a true multi-objective GA for approximating the Pareto front in such a high-dimensional space (see Table 1) one could "scalarize" the problem by assigning weights to each of those criteria in order to obtain a single objective function. The problem is then how to assign those weights in a way that a pleasant-looking lay-out emerges. In the graph lay-out problem it is observed that different sets of weights lead to different final solutions which display different aspects/properties of the graph at hand. The final choice is thus often a matter of personal preference.

3 INTERACTIVE EVOLUTIONARY COMPUTATION

When the fitness function is subjective, an interactive GA is implemented in a way that the quality of every candidate solution has to be externally provided by the user to the GA which then performs recombination and/or mutation of individuals in the current population in order to generate the next population. The new individuals must then be presented to the user which has the task to evaluate them according to his/hers preferences.

Caldwell and Johnston[21] have developed an IGA which allows for the interactive evolution of a face of a suspect seen by the user at the scene of a crime. An IGA has been applied by Louis and Tang[22] to the traveling salesman problem (TSP). First the user visually decomposes the TSP into clusters of cities and then a GA is applied to solve the TSP corresponding to each cluster. Finally, the user either (i) manually connects sub-tours in order to get a complete solution or (ii) defines a promising region between two clusters and then let the system perform an exhaustive search among edges to delete and add in this region such that the total tour length is minimized.

It is interesting to note that when the evolved artifact is represented in two- or three-dimensional space the individuals can be *simultaneously* presented (in small numbers though) to the user which can then rank them by comparison. However, for a musical solution, say a jazz solo, which develops itself in time, the user has to hear them carefully, one at a time, and then proceed to the ranking stage.

It has been observed that the user's task often leads to fatigue and that an IGA should be carefully implemented if it is to be useful. One possible way to remove the human being from the IGA loop and thus speed up the evolutionary process is to substitute it by a neural network trained to reflect the user's preferences. That has been tried by Biles[9] in order to simplify his original procedure which consisted in presenting each solution (a jazz solo) to an audience for evaluation. Unfortunely good results were not obtained indicating that this option is also hard. In the work by Johanson and Poli[11] an interactive system allows users to evolve short musical sequences using interactive GP. As the user is the bottleneck in the process, the system takes rating data from a user's run and uses it to train a neural network based automatic rater which can then replace the user and allow for longer runs. The best pieces generated by the automatic raters were reported to be pleasant but were not, in general, as nice as those generated in user interactive runs.

An approach to the interactive development of programs for image enhancement using GP is presented by Poli and Cagnoni[23]. There is no fitness function and the user decides which individual should be the winner in the tournament selection. Good solutions to real-life problems are reported after only hundreds of evaluations. Also, a strategy to reduce user interaction is proposed: the choices made by the user in interactive runs are recorded and later used to build a model which can replace the user in longer runs.

There are other situations of scientific and technological importance where, although a reasonable fitness function can be designed, the evolutionary process would be improved if any additional user knowledge (perhaps hard to code into the fitness function) could be used in the evaluation or ranking of the population. In this case, one would profit from a *supervised* process where the user monitors the evolution and occasionally intervenes in order to guide the process towards what is perceived as a better region of the search space at least according to the user experience. An example of such approach has been presented recently by Boscheti[24], where IGAs are proposed for a class of geophysical inversion problems. For additional IEC applications and a survey, the reader is referred to Takagi[25, 26] and references therein.

As already mentioned, an important area where user preferences are essential is that of MOO problems, and we believe that the idea of co-evolution can bring some needed help to the design of an IGA for this class of problem.

4 CO-EVOLUTION

In a standard GA the fitness function does not change with time so that the population can be thought of as climbing a fitness landscape and gradually converging to one of its peaks. In contrast to these static landscapes, natural evolution happens in a dynamic fitness landscape where organisms are constantly adapting to each other and to their environment.

Artificial co-evolutionary algorithms have been used in the solution of practical problems and one idea is that of *cooperation*. Potter and De Jong[27] used it for function optimization in R^n where n subpopulations are maintained and the fitness of a given individual of a particular subpopulation is an estimate of how well it cooperates with other subpopulations to produce good solutions. These ideas were later applied to the co-evolution of neural networks[28] and sets of rules[29].

Another approach is that of *competition*. In the pioneering work of Hillis[30] it was applied to the problem of evolving minimal sorting networks for lists of a given number of elements. It was observed that the co-evolution of test cases along with the sorting networks results in an improved procedure. Two independent populations are maintained: one of sorting networks (or hosts) and the other of test cases (parasites). The fitness of each sorting network was measured by its ability to correctly solve test cases while the fitness of each test case was proportional to the number of times it was incorrectly sorted by the networks. Both populations evolved simultaneously, interacting through the fitness function.

Paredis[31] used a co-evolutionary approach to improve the genetic evolution of neural networks. Again two populations were maintained: one of neural networks and another of examples of the classification task which is submitted to the neural networks. The fitness of a neural network is defined as the number of successful classifications of the twenty most recently encountered examples. On the other hand, the fitness of an example is the number of times it was incorrectly classified by the neural networks it encountered most recently. However, the example population consists all the time of the same 200 pre-classified examples. To evolve decision trees, Siegel[32] also performed competitive co-evolution with fixed training examples.

Rosin and Belew[33] used competitive co-evolution for game-learning problems in which the fitness of an individual in a host population is based on a direct competition with individual(s) from a parasite population.

Inspired by the work of Hillis[30], a co-evolutionary GA has been proposed by Barbosa[34] in which two evolving populations are used to solve min-max problems. Several successful small scale applications were reported in [34]. Later[35] that approach was applied to a class of optimization problems stated as an adversary game between two players ("nature" and the designer), as well as to constrained optimization problems[36].

Competitive co-evolution was also used by Sebald[37] in the min-max design of neural net controllers for uncertain plants while Husbands[38] used a cooperative/competitive multi-population distributed co-evolutionary GA to handle a generalized version of job shop scheduling.

In this paper an interactive, co-evolutionary GA is proposed for MOO problems where the user's preferences are subjective, such as in the graph lay-out problem considered here. However, the procedure can not be classified neither as a competitive nor as a cooperative co-evolutionary GA.

5 THE CO-EVOLUTIONARY GA

In the co-evolutionary GA proposed here, two populations will be maintained: a graph lay-out population and a weight-set population. The lay-out population is composed of individuals that contain the coordinates of all vertices in the graph while the weights population is composed of individuals that contain, each one, a set of weights. An independent GA is applied to each population and the coupling of the evolutionary processes is made through the fitness evaluation. Each one of the metrics/objectives can be exactly computed for a given lay-out but the fitness of this lay-out depends on how – i.e. with which set of weights – those objectives are linearly combined.

A fitness function must also be defined for the weight population. Here it will be defined as follows. After the lay-out population evolves for a given number of generations (with its fitness being computed according to a fixed set of weights) a sample of lay-outs from the current population is displayed for user inspection. The user then ranks them according to his/hers personal preferences, which may be different from their current ranking in the population.

Now it is time for the weight population to evolve while the current lay-out population is kept "frozen". The fitness function value of a given set of weights is then computed as follows. Each individual (set of weights) evaluates the sample of lay-outs and produces its own ranking. The fitness of a given set of weights is higher, the closer its ranking approaches the ranking provided by the user. A set of weights that ranks the sample in a very different order from that chosen by the user is not very useful and has thus a low fitness.

The weight population is now evolved in order to search for that set of weights which produces a ranking the most closely resembling the one provided by the user. During a fixed number of generations, the lay-out population is kept "frozen".

After a (hopefully) better set of weights is found, the lay-out population is then allowed to evolve with its fitness evaluation being now performed using the newly found (best) set of weights.

The process is then repeated for a given number of cycles until a satisfactory graph lay-out emerges.

As a result, the fitness of an element in the lay-out population depends on the current set of weights, which depends on the evolution of the weight population whose fitness, in turn, depend on how well the current set of weights reflects the user's personal preferences concerning the graph lay-outs. In other words, the evolution of the lay-out population happens in a fitness landscape that changes every time a new set of weights is presented by the weights population. On the other hand, the fitness landscape of the weights population changes every time *the user* presents its ranking of a sample of the current lay-out population. Figure 1 displays the pseudo-code for the algorithm.

In order to minimize fatigue, the user only points and clicks the mouse on *some* of the displayed lay-outs following a decreasing order of his/her preference. The remaining lay-outs are ranked after those picked by the user and maintain their relative order.

```
Algorithm
    Initialize lay-out population randomly
    Initialize weight-set population randomly
    Compute each criterion for all graph lay-outs
    while not(user satisfied) do
        Display a sample from the lay-out population
        The user ranks the sample
        for $i = 1, 2, ..., max\_gen\_w$ do
            Evaluate population of weights
            Generate new population of weights
        Pick the best set of weights
        for $j = 1, 2, ..., max\_gen\_l$ do
            Compute each criterion for all graph lay-outs
            Evaluate population of lay-outs
            Generate new population of lay-outs
end
```

Figure 1: Pseudo-code for the algorithm.

Details of the GA used for each population are given next.

5.1 The genetic algorithm for the weights population

As suggested by an anonymous reviewer, a system of linear inequalities could be written defining the set of all feasible sets of weights, that is, weights that reproduce a given user's ranking. However, it cannot be proven that such a set is always non-empty: there is a human making choices. As a result, a generational GA with a rank-based selection scheme was adopted for the evolution of the weights population, where each chromosome is simply a vector of positive real numbers (the weights) whose sum is 1.

The fitness of a set of weights is evaluated by its capacity to rank the lay-out population in accordance with the user's preferences. The process can be explained as follows. When the user inputs his/her own rank, it can be seen as a vector U describing a specific ordering. The fitness of a set of weights S can be calculated by:

$$f(S) = \sum_{j=1}^{m} C_j (W_j - U_j)$$

where W is a vector like U, but describing the ordering generated by the individual S. C is a fixed vector of constants which allows one to emphasize the importance of the first individuals. In our tests, $m = 9$ and the elements of C are $\{9, 8, 7, 6, 5, 4, 3, 2, 1\}$.

The crossover operator used was the blend crossover - BLX[39] which generates one offspring by randomly generating for each weight a value in the range $[w_i^s - \delta, w_i^l + \delta]$ with $\delta = \alpha(w_i^l - w_i^s)$ and where w_i^s and w_i^l are the smallest and largest values respectively for w_i in the parents' chromosomes (α was set to 0.3).

Two mutation schemes are used, which will be referred to as weak perturbation mutation - WP and strong perturbation mutation - SP. First of all, a value is randomly chosen from the interval [0,1]. If it is lower than 0.85, mutation will occur. Next, the WP mutation will be applied with a probability of 0.7 and the SP will occur with probability of 0.3 (1.0 - 0.7). The mutation operators work as follows: (i) WP: One weight w_i of the set is randomly chosen and perturbed by an amount limited to the interval $[-0.3w_i, 0.3w_i]$; (ii) SP: One gene (a weight) is randomly chosen and has its value changed to another value randomly picked in the interval [0,1].

5.2 The genetic algorithm for the lay-out population

The GA used here is a standard generational one where each candidate solution is encoded as a real vector containing the coordinates (x_i, y_i) of each vertex of the graph. The initial population is randomly generated and the selection process is rank-based.

The GA for graph lay-out tries to optimize for several aesthetic criteria. One of them is the energy function defined by Kamada and Kawai[40]

$$\mathcal{E} = \sum_{i=1}^{|V|-1} \sum_{j=i+1}^{|V|} \frac{1}{2} k_{ij} (|p_i - p_j| - l_{ij})^2 \qquad (1)$$

where $|V|$ is the number of vertices, p_i is the position vector of vertex i, and k_{ij} and l_{ij} are respectively the spring force and the ideal distance between vertices i and j. The idea is to make the Euclidean distance between two vertices as close as possible to the ideal "graph theoretic" distance, which is defined as $l_{ij} = Ld_{ij}$ where d_{ij} is the distance between vertices i and j measured as the length of the shortest path between them and L is the ideal length of an edge in the graph, given by $L = L_0 / \max\{d_{ij}\}$ where L_0 is the length of the side of the square display area. Finally, the spring constant in Eq. (1) is given by $k_{ij} = K/d_{ij}^2$ where K is a constant. Due to the observation that this energy function, which often leads to uniform edge length and uniform vertex distribution, is not always able to achieve a good layout, several other criteria have also been introduced. The objective function is thus composed by a weighted sum of those criteria and the energy function $\mathcal{E}$. All criteria adopted are listed in Table 1 with the complexity associated with their computation, where $|E|$ denotes the number of edges in the graph.

Only mutation operators are used in this GA. However, the approach adopted here is different from that of the standard GA; here, each *individual* has a probability of 0.85 to be submitted to mutation. The first mutation operator is the single vertex mutation – SV, which perturbs the coordinates of a randomly chosen vertex by an amount not larger than L. The second one is the exchange vertex mutation – EV, which exchanges two randomly pre-selected vertices of the same graph. Finally, a third one, called vertex replacement mutation - VR was also defined. Just like the SP mutation of the GA for the weight population, it simply replaces one gene for another value randomly chosen but feasible, i.e., the (x, y) coordinates of a vertex are replaced by a pair of values in the interval $[0, L_0]$. The relative probability of these 3 operators was set as 0.5, 0.2 and 0.3, respectively. One should note that both GA's have to adapt to – potentially drastic – landscape changes, so it seems a good idea to have a high mutation probability.

The reason for not using a crossover operator is the fact that the schemes tested not only did not bring any improvement but also made the process slower. This seems to be due to the difficulty known as the *competing conventions problem* in the area of evolutionary neural networks. In our case, it happens when two (or more) identical layouts of the same graph are either shifted, rotated or inverted. It is desirable for a crossover operator that, when combining two equivalent layouts, the offspring generated be similar to its parents. We intend to develop such an operator, using ideas from [41].

6 NUMERICAL EXPERIMENTS

In order to illustrate the feasibility and performance of the approach proposed here, the task of laying out a simple planar graph was undertaken.

Criterion	Abb.	Description	Complexity
Energy	En	Potential energy $\mathcal{E}$	$\Theta(\|V\|^2)$ [a]
Edge Crossings	Ec	Number of intersections between edges	$\Theta(\|E\|^2)$
Edge Deviation	Ed	Difference between edge lengths	$\Theta(\|E\|^2)$
Edge attraction	Ea	Distance between connected vertices	$\Theta(\|E\|)$
Vertices Repulsion	Vr	Inverse of distance between vertices	$\Theta(\|V\|^2)$
Centripetal Attraction	Ca	Distance between vertices and the centroid of lay-out	$\Theta(\|V\|)$
Central Uniformity	Cu	Difference between vertices distances to the centroid of lay-out	$\Theta(\|V\|^2)$
Vertex-edge Distance	VEd	Inverse of distance between vertices and edges	$\Theta(\|V\|\|E\|)$

---

[a] Assuming that the d_{ij} are previously calculated and stored.

Table 1: Aesthetics criteria adopted in the algorithm

Figure 2 shows 9 elements from the initial population which were presented to the user.

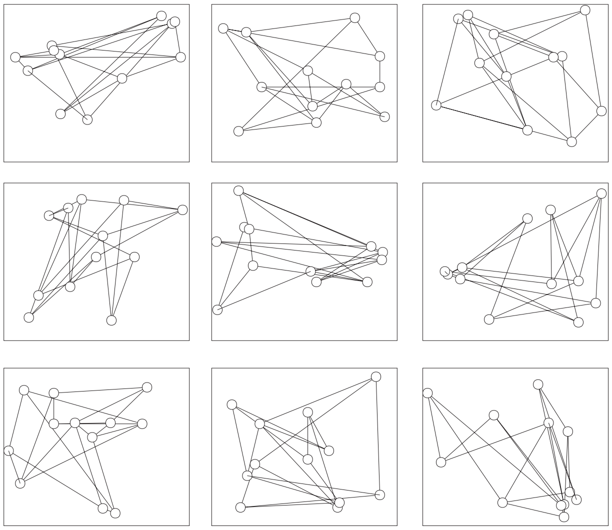


Figure 2: A sample from the initial population.

The first run was made by a user who did not care about the number of edge-crossings in the drawing.

The weights associated with the criteria used in this case, En, Ed, Ec, Ca and Vr then evolved respectively to 0.5394, 0.2854, 0.06554, 0.050776 and 0.05887 and a solution with many edge crossings was obtained but with an interesting spatial structure displayed as shown in Figure 3.

The second run was made by an user who wanted a solution with a minimum of edge-crossings. As expected, the weights evolved to quite different values: 0.0008176, 0, 0.6478, 0.02042 and 0.3310. It is easy to see that the (third) weight associated with edge-crossings now has evolved to a value ten times higher than that of the previous case and the one associated with the energy is almost $1/1000$ of the previous solution, indicating that this criteria goes “against” a planar solution in this problem. The graph obtained has no edge-crossings and is displayed in Figure 4.

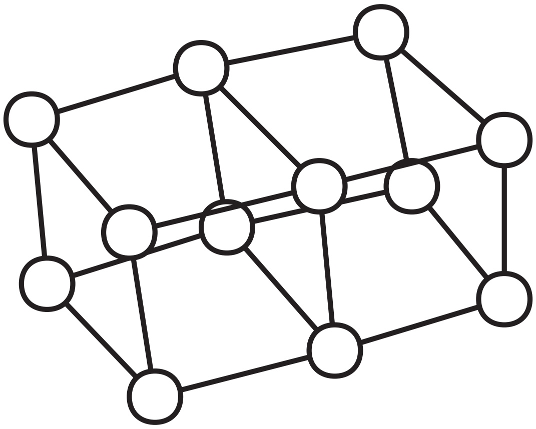


Figure 3: Solution for user 1.

7 CONCLUSIONS

The work presented here reports on an investigation being conducted concerning the design of an interactive co-evolutionary genetic algorithm for multi-objective optimization problems and its application to the graph lay-out problem. Comparing with previous interactive evolutionary algorithms, the proposed interactive co-evolutionary GA can use a larger population for the application at hand since the user only has to rank a sample from the corresponding population. A single paradigm – the genetic algorithm – is used for both tasks, namely, that of searching for a good solution of the multi-objective optimization problem (a graph lay-out, in the present application) and that of learning the user’s subjective preferences.

Acknowledgements

This paper was written while the first author was visiting the Colorado State Artificial Intelligence Laboratory. The hospitality provided by Prof. Darrel Whitley is gratefully acknowledged. The authors would also like to thank the reviewers for comments and suggestions which helped improve the quality of the paper.

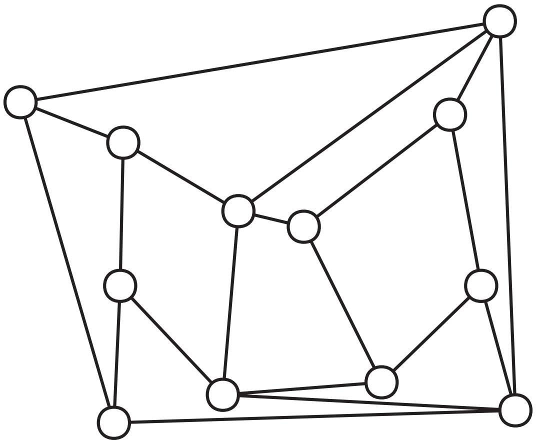


Figure 4: Solution for user 2.

References

[1] R. Dawkins. *The Blind Watchmaker*. Longman, Essex, 1986.

[2] K. Sims. Artificial evolution for computer graphics. *Computer Graphics*, 25(4):319–328, July 1991.

[3] K. Sims. Evolving 3D morphology and behaviour by competition. In R. A. Brooks and P. Maes, editors, *Artificial Life IV*, pages 29–39. MIT Press Cambridge, MA, 1994.

[4] M. Herdy. Evolution strategies with subjective selection. In W. Ebeling, H.-P. Schwefel, and H.-M. Voigt, editors, *Proc. of the Parallel Problem Solving from Nature - PPSN IV*, pages 22–31, Berlin: Springer, 1996.

[5] M. Herdy. Evolutionary optimization based on subjective selection - evolving blends of coffee. In *Proc. of the 5th European Congress on Intelligent Techniques and Soft Computing, EUFIT '97*, pages 640–644, 1997.

[6] P.J. Bentley, editor. *Evolutionary Design by Computers*. Academic Press Ltd., London, 1999.

[7] A. Horner and D. Goldberg. Genetic algorithms and computer-assisted music composition. In Richard K. Belew and Lashon B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms and their Applications*. Morgan Kaufmann, San Mateo, CA, 1991.

[8] J.A. Biles. GenJam: A genetic algorithm for generating Jazz solos. In *International Computer Music Conference*, San Francisco, CA, 1994.

[9] J.A. Biles. Neural network fitness functions for a musical IGA. In *Soft Computing Conference*, 1996.

[10] B.L. Jacob. Composing with genetic algorithms. In *International Computer Music Conference*, San Francisco, CA, 1995.

[11] B.E. Johanson and R. Poli. GP-music: An interactive genetic programming system for music generation with automated fitness raters. Technical Report CSRP-98-13, University of Birmingham, School of Computer Science, May 1998.

[12] F. Gruau and K. Quatramaran. Cellular encoding for interactive evolutionary robotics. Technical Report Cognitive Science Research Paper CSRP425, School of Cognitive and Computing Sciences, University of Sussex, Brighton BN1 9QH, England, UK, 1996.

[13] J.D. Schaffer. *Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, Nashville, TN, 1984.

[14] J.D. Schaffer. "Multiple objective optimization with vector evaluated Genetic Algorithms". In *Proc. of the 1st Int. Conf. on Genetic Algorithms*, pages pp. 93–100, 1985.

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

[16] Jeffrey Horn. *Multicriterion Decision Making*, volume 1, pages F1.9:1 – F1.9:15. IOP Publishing Ltd. and Oxford University Press, 1997.

[17] D.A. Van Veldhuizen and G.B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1998.

[18] C.A.C. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, 1999.

[19] M. Shibuya, H. Kita, and S. Kobayashi. Integration of multi-objective and interactive genetic algorithms and its application to animation design. In *Proc. of IEEE Systems, Man, and Cybernetics*, volume III, pages 646–651, 1999.

[20] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. Annotated bibliography on graph drawing algorithms. *Computational Geometry: Theory and Applications*, 4(5):235–282, 1994.

[21] C. Caldwell and V.S. Johnston. Tracking a criminal suspect through face-space with a genetic algorithm. In Richard K. Belew and Lashon B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms and their Applications*. Morgan Kaufmann, San Mateo, CA, 1991.

[22] S.J. Louis and R. Tang. Interactive genetic algorithms for the traveling salesman problem. In *Proc. of the Genetic and Evolutionary Computation Conference*, 1999.

[23] R. Poli and S. Cagnoni. Evolution of pseudo-colouring algorithms for image enhancement with interactive genetic programming. Technical Report CSRP-97-5, School of Computer Science, The University of Birmingham, B15 2TT, UK, January 1997.

[24] F. Boschetti and L. Moresi. Comparison between interactive (subjective) and traditional (numerical) inversion by genetic algorithms. In *2000 Congress on Evolutionary Computation*, pages 522–528, San Diego, CA, USA, July 2000. IEEE Service Center.

[25] H. Takagi. Interactive evolutionary computation: System optimization based on human subjective evaluation. In *IEEE Int'l Conf. on Intelligent Engineering Systems (INES'98)*, pages 1–6, Vienna, Austria, Sept. 1998.

[26] H. Takagi. Interactive evolutionary computation - Cooperation of computational intelligence and human *kansei*. In *5th Int'l Conf. on Soft Computing (IIZUKA'98)*, pages 41–50. World Scientific, Iizuka, Fukuoka, Japan, Oct. 1998.

[27] M.A. Potter and K.A. De Jong. A cooperative co-evolutionary approach to function optimization. In *Third Parallel Problem Solving from Nature*, Jerusalem, Israel, 1994.

[28] M.A. Potter and K.A. De Jong. Evolving neural networks with collaborative species. In *Proc. of the 1995 Summer Computer Simulation Conference*, Ottawa, Ontario, Canada, 24–26 July 1995.

[29] K.A. De Jong and M.A. Potter. Evolving complex structures via cooperative coevolution. In *Fourth Annual Conference on Evolutionary Computation*, San Diego, CA, 1–3 March 1995.

[30] W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[31] J. Paredis. Steps towards co-evolutionary classification neural networks. In R. Brooks and P. Maes, editors, *Artificial Life IV*. MIT Press/Bradford Books, 1994.

[32] E.V. Siegel. Competitively evolving decision trees against fixed training cases for natural language processing. In K. Kinnear, editor, *Advances in Genetic Programming*. MIT Press Cambridge, MA, 1994.

[33] C.D. Rosin and R.K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In L.J. Eshelman, editor, *Proc. of the Sixth International Conference on Genetic Algorithms and their Applications*, Pittsburgh, PA, July 1995.

[34] H.J.C. Barbosa. A genetic algorithm for min-max problems. In E.D. Goodman, V.L. Uskov, and W.F. Punch III, editors, *EvCA'96 - First Int. Conf. on Evolutionary Computation and its Applications*, pages 99–109, Moscow, 24–27 June 1996. Institute of High Performance Computer Systems of the Russian Academy of Sciences.

[35] H.J.C. Barbosa. A coevolutionary genetic algorithm for a game approach to structural optimization. In Th. Baeck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms and their Applications*, pages 545–552, East Lansing, MI, July 1997.

[36] H.J.C. Barbosa. A coevolutionary genetic algorithm for constrained optimization problems. In *Proc. of the 1999 Congress on Evolutionary Computation*, pages 1605–1611, Washington, DC, USA,, July 1999. IEEE Service Center.

[37] A.V. Sebald and J. Schlenzig. Minimax design of neural net controllers for highly uncertain plants. *IEEE Transactions on Neural Networks*, 5(1):73–82, 1994.

[38] P. Husbands. Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In T. Fogarty, editor, *Evolutionary Computing, Lecture Notes in Computer Science*, volume 865, pages 150–165. Springer-Verlag, 1994.

[39] L.J. Eshelman and J.D. Schaffer. Real coded genetic algorithms and interval schemata. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, San Mateo, CA, 1993.

[40] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, April 1989.

[41] Jürgen Branke, Frank Bucher, and Hartmut Schmeck. A genetic algorithm for drawing undirected graphs. In *Proc. of the Third Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, pages 193–206, 1997.

Potholes on the Royal Road

Theodore C. Belding
Center for the Study of Complex Systems
University of Michigan
Ann Arbor, MI 48109-1120 USA
`Ted.Belding@umich.edu`
`http://www-personal.umich.edu/~streak/`

Abstract

It is still unclear how an evolutionary algorithm (EA) searches a fitness landscape, and on what fitness landscapes a particular EA will do well. The validity of the building-block hypothesis, a major tenet of traditional genetic algorithm theory, remains controversial despite its continued use to justify claims about EAs. This paper outlines a research program to begin to answer some of these open questions, by extending the work done in the royal road project. The short-term goal is to find a simple class of functions which the simple genetic algorithm optimizes better than other optimization methods, such as hillclimbers. A dialectical heuristic for searching for such a class is introduced. As an example of using the heuristic, the simple genetic algorithm is compared with a set of hillclimbers on a simple subset of the hyperplane-defined functions, the pothole functions.

1 BACKGROUND

Evolutionary algorithms (EAs) are computational search methods based on biological evolution. Some common EAs are *genetic algorithms* (GAs) [12, 9, 18], *evolutionary programming* (EP) [6], *evolution strategies* (ESs) [24], *genetic programming* (GP) [17], and *classifier systems* [14]. The study of EAs is called *evolutionary computation* (EC).

EAs are increasingly important in such areas as function optimization, machine learning, and modeling. However, as Mitchell et al. emphasized in the royal road (RR) papers [19, 7, 20], it is still unclear how an EA searches a fitness landscape, or even what an EA *does*. It is also unknown what types of problem are easy or hard for a particular EA, how various landscape features affect problem difficulty for an EA, or under what circumstances an EA will outperform another search method. Even less work has been done to classify the features of real-world problems that may be relevant to EA performance. Moreover, the selection of EA parameters such as mutation rate or population size is still largely a black art, despite some promising research in this area. This lack of theory makes the selection and configuration of an EA for a given problem difficult.

A major open question in EC is the function and importance of the *crossover* operator, which recombines two individuals. Holland [12] has argued that crossover is central to an EA's efficacy. The theoretical basis for this is the *building-block hypothesis* (BBH) [9], which states that an EA uses crossover to repeatedly combine compact subsolutions with high observed fitness from different individuals, forming more complete subsolutions with even higher observed fitness. Such subsolutions are called *building blocks*. When an EA uses crossover on symbol strings from the set A^ℓ, where A is the set of possible symbols and ℓ is the length of a string, the building blocks are short *schemata* with high observed fitness. Schemata are members of the set $(A \cup \{*\})^\ell$, where $*$ is a wildcard symbol. They are hyperplanes in the search space. Holland's schema theorem [12] implies that short schemata with consistently above-average observed fitness tend to increase exponentially in frequency over several generations. (If operators other than reproduction are neglected, this is true for all partitions of the search space, not just for a partition into schemata. However, applying crossover to symbol strings induces a natural partition of the space into schemata [27]. Furthermore, short schemata are preserved by crossover. This makes schemata particularly relevant when studying EAs that use crossover on strings.) Applying crossover to individuals with high fitness is a plausible heuristic for generating offspring that will also be highly fit. The chance that this heuristic succeeds can be quantified using Price's covariance and selection theorem [1]. Implicit in the BBH is also the hypothesis that there are many real-world problems amenable to solution by this process.

A common misconception is that a schema has a unique, well-defined fitness, which is the average fitness of all of its possible instances, and that observed fitnesses are estimates of these "actual" values. In general, no such unique schema fitness exists, and the schema theorem makes no such assumption [11]: The observed fitness of a schema is the average fitness of its instances in the current population. This value depends on the distribution of schemata in the current population, which is biased by the EA over time. A uniform distribution is only seen immediately after generating an initial random population, if ever. Hence, there is no justification for arbitrarily defining a schema's fitness to be the average over a uniform distribution. A schema may be a highly-fit building block in one population but not in another, even under the same fitness function. Grefenstette [10] made essentially the same point when he criticized the "static building-block hypothesis".

The BBH is often used to explain how EAs work and to justify the importance of crossover. However, there is no theory that specifies in detail the conditions necessary for the BBH to be valid and thus for crossover to be beneficial. While there is empirical evidence in favor of the BBH [13], its validity in general for the SGA and for other EAs using crossover remains controversial [6], and the schema theorem's relevance to EA theory has been questioned as well [27, 28]. In particular, uniform crossover, which is more likely to break up short building blocks than traditional crossover, is very effective on some problems [26], and it may be that on some problems crossover acts as a macromutation operator, rather than as an operator that recombines building blocks [16]. More generally, it is not clear how to formulate a BBH that is valid for an arbitrary EA operating on an arbitrary representation of solutions [22].

2 RESEARCH PROGRAM

This paper presents a research program to extend the RR papers [19, 7, 20] in testing the validity of the BBH, focusing specifically on the *simple genetic algorithm* (SGA) [9]. The SGA is a GA that uses fitness-proportionate selection, single-point crossover, and point mutation to evolve a single panmictic population of bit strings, with each generation completely replacing the previous one. I focus on the SGA because it is a relatively simple EA with a large theoretical literature and because many EAs descend directly or indirectly from it. A theory developed for the SGA has a relatively good chance of being applicable to other EAs.

As in the RR papers, I use function optimization to compare the SGA with other search algorithms. I do this because it is an increasingly important application for EAs, with relatively clear performance criteria, and because a simple fitness function is easy to design and implement. Also, function optimization can be viewed as search, so theories developed for it may be relevant to other applications of search, for instance artificial intelligence [21] and evolutionary biology [30].

As De Jong [2] pointed out, the SGA is not a function optimizer, per se. But if the BBH is valid, the SGA should work particularly well as an optimizer on functions rich in building blocks that can be recombined to reach the optimum. Hence, to determine the validity of the BBH, it would be useful to know the class of functions on which the SGA outperforms other optimizers. Examining what makes this class of functions particularly easy for the SGA will also help us to predict which functions the SGA will perform well on. One step towards this goal is to find a simple class of functions on which the SGA outperforms hillclimbers. This was the goal of the RR papers and is also the immediate goal here. To meet this goal, the SGA should consistently perform extremely well on the functions and outperform hillclimbers by a wide margin, for a reasonable set of performance criteria. (Note that this is a different question from that addressed by Wolpert and Macready's [29] no free lunch (NFL) theorem.)

This paper takes a different approach from that of the RR papers, although the ultimate goal remains the same. In those papers, Forrest and Mitchell [7] determined that *hitchhiking* was a major factor limiting SGA performance on the RR functions, causing it to perform worse than the random mutation hillclimber (RMHC). Hitchhiking occurs when detrimental or neutral alleles increase in frequency due to the presence of nearby beneficial alleles on the same chromosome [8]. This can cause beneficial alleles at the same loci as the hitchhiking alleles to die out in the population, preventing the SGA from finding any highly-fit individuals that contain those alleles. (The fact that schemata do not have unique, well defined fitnesses is a necessary precondition for hitchhiking.) After identifying this problem, Mitchell et al. [20] investigated how to make the SGA perform more like an *idealized genetic algorithm* that was unaffected by hitchhiking. They developed the RR function $R4$, which reduced hitchhiking by lowering the fitness jump from one level of the function to the next. In effect, they made the SGA outperform the RMHC by making the functions easier for the SGA. In contrast, I attempt to make them harder for hillclimbers. The RR functions are easy for hillclimbers like the RMHC because they are convex: An algorithm never needs to go downhill in order to reach the global optimum. To make these functions hard for hillclimbers, I add *potholes* to them: valleys in the fitness landscape that block a hillclimber's path to the optimum [19]. This produces the *pothole functions*, described in Section 3. (Holland proposed a class of RR functions, described by Jones [15], that also contained potholes.)

Pothole functions are a very simple subset of Holland's hyperplane-defined functions (HDFs) [13]; potholes are examples of HDF *refinements.* A long-term goal is to design a class of pothole functions with parameters that can be varied to select the landscape features present in a function, as well as its overall difficulty. An arbitrary number of functions could then be generated with the desired characteristics by using those parameters to define probability distributions, which in turn could be used to choose the schemata that contribute to an individual's fitness, along with their fitness contributions. (In this paper, these will be called a function's *significant schemata.*) Such a class would allow a researcher to use statistical methods to calculate the certainty of statements about an algorithm's performance across the entire class, while being easier to understand than more general classes of HDFs. However, it is not yet clear what parameters or distributions should be used, if the goal is to describe a class of functions that is easy for the SGA yet hard for hillclimbers. The immediate goal is to use hand-designed pothole functions as testbeds to determine what regions of distribution space should be used for randomly-generated functions.

I base my work on the RR functions because they were explicitly designed to investigate the validity of the BBH by studying the SGA's performance on functions rich in building blocks. The significant schemata in a RR function are not building blocks in every population, since their fitness depends on the current population. However, the functions are defined so that they are building blocks in all contexts except in the occurrence of hitchhiking, since they make only positive fitness contributions. The functions are "rich in building blocks" in this sense; the pothole function p_1 described in Section 3 has the same characteristic. In this paper, a schema that makes a positive fitness contribution (ignoring the fitness contribution of other schemata) will be called a *beneficial schema.* (It is possible to define a building block as any beneficial schema, in contrast to the definition given earlier. This definition is related to Fisher's [5] *average excess,* but it makes the relationship between the BBH and the schema theorem less clear [J. H. Holland, personal communication].) Like the RR functions, the pothole functions are not meant to be realistic. Since the fitness contribution of every schema is specified in advance, schemata can be used as tracers: They can be related to individual landscape features, and their frequency in the EA population can be tracked over time [19]. Hypotheses about the effects of various landscape features on EA behavior can then be formulated and tested. This knowledge can then be applied to the study of real-world functions.

1. Create a function that is easy for the SGA, for some performance criteria.
2. Use domain-specific knowledge to design a simple algorithm that is able to optimize that function better than the SGA. If no such algorithm can be found, or if all such algorithms incorporate unreasonable amounts of domain-specific knowledge, go to Step 4.
3. Modify the function so that it is hard for the simple algorithm yet still easy for the SGA. If no such function can be found, go back to Step 1 and start over. Otherwise, go back to Step 2.
4. Stop — a candidate function has been found.

Algorithm 1: A dialectical heuristic for finding a simple function that is easy for the SGA but hard for other optimizers. (Note that this heuristic may never succeed.)

Given enough domain-specific knowledge, it is plausible that a specialized optimization method can be designed to outperform the SGA on *any* sufficiently restricted class of functions. (The NFL theorem does not hold if the subset of functions being considered has measure 0 in distribution; this is true for many subsets of interest, in particular all countable subsets [J. H. Holland, personal communication].) Therefore, the issue is not whether the SGA will outperform *all* other algorithms on a given class. Rather, it is: How much domain-specific knowledge is it reasonable to incorporate into an algorithm before it becomes over-specialized or too expensive to design and implement, outweighing any performance advantage over the SGA? A related question is: How broad must a class of functions be before the SGA outperforms a specialized optimizer on it? More generally, the RR papers suggest a *dialectical heuristic* to search for a simple function that is easy for the SGA but hard for hillclimbers (Algorithm 1). (Note that "dialectic" here simply denotes "the existence or working of opposing forces" [25].) While this heuristic is very straightforward, it has never been articulated explicitly. Since it is a heuristic, it may never succeed; however it is a plausible way to search for the desired class of functions.

The remainder of this paper provides a concrete example of using this heuristic. The pothole function p_1 is introduced and shown to be difficult for the RMHC but easy for the SGA. Then a variant of the RMHC, the lax random-mutation hillclimber (LRMHC), is defined, which knows the depth of the potholes in p_1 and is able to cross over them to reach the optimum. This hillclimber is shown to outperform the SGA on p_1. The paper concludes with a discussion of the results.

Table 1: The pothole function p_1. An individual's fitness is calculated by summing the fitness contributions of the schemata of which the individual is an instance, and then adding this sum to a base fitness of 100. If the result is less than 0, it is reset to 0. The global optimum is a string of 64 1s, with a net fitness of 115.

Level	Schema		Fitness
1	s_0	`11111111********************************************************`	1.0
	s_1	`********11111111************************************************`	1.0
	s_2	`****************11111111****************************************`	1.0
	s_3	`************************11111111********************************`	1.0
	s_4	`********************************11111111************************`	1.0
	s_5	`****************************************11111111****************`	1.0
	s_6	`************************************************11111111********`	1.0
	s_7	`********************************************************11111111`	1.0
2	s_8	`1111111111111111************************************************`	1.4
	s_9	`****************1111111111111111********************************`	1.4
	s_{10}	`********************************1111111111111111****************`	1.4
	s_{11}	`************************************************1111111111111111`	1.4
3	s_{12}	`11111111111111111111111111111111********************************`	1.0
	s_{13}	`********************************11111111111111111111111111111111`	1.0
4	s_{14}	`1111111111111111111111111111111111111111111111111111111111111111`	1.0
	s_{15}	`111111111*******************************************************`	−0.1
	s_{16}	`11111111*1******************************************************`	−0.1
	s_{17}	`******1*11111111************************************************`	−0.1
	s_{18}	`*******111111111************************************************`	−0.1
	s_{19}	`****************111111111***************************************`	−0.1
	s_{20}	`****************11111111*1**************************************`	−0.1
	s_{21}	`**********************1*11111111********************************`	−0.1
	s_{22}	`***********************111111111********************************`	−0.1
	s_{23}	`********************************111111111***********************`	−0.1
	s_{24}	`********************************11111111*1**********************`	−0.1
	s_{25}	`**************************************1*11111111****************`	−0.1
	s_{26}	`***************************************111111111****************`	−0.1
	s_{27}	`************************************************111111111*******`	−0.1
	s_{28}	`************************************************11111111*1******`	−0.1
	s_{29}	`******************************************************1*11111111`	−0.1
	s_{30}	`*******************************************************111111111`	−0.1

3 POTHOLE FUNCTIONS

Following the dialectical heuristic described in Section 2, I modified the RR functions to make them harder for simple hillclimbers by adding *potholes.* Potholes are detrimental schemata that contain beneficial schemata, and which, in turn, are necessary to reach beneficial schemata with higher fitness contributions [19]. This produces the class of *pothole functions.* All experiments in this paper were performed on the pothole function p_1, which is defined in Table 1. That table lists all of the schemata that contribute to an individual's fitness, along with their fitness contributions; these schemata are called the function's *significant schemata.*

The fitness $p_1(x)$ of a string $x \in \{0,1\}^{64}$ is given by

$$p_1(x) = \max\left\{0,\ 100 + \sum_{s \in S \mid x \in s} \mu(s)\right\}, \qquad (1)$$

where S is the set of significant schemata for p_1, s is a schema in S, and $\mu(s)$ is the fitness contribution of s. The notation $x \in s$ stands for "the string x is an instance of the schema s". Individuals have a base fitness of 100, so that in other pothole functions they may be less fit than the base fitness without having a negative fitness; the fitness is forced to be equal or greater than 0 so that fitness-proportionate selection may be used. The global optimum is a string of 64 1s, which has a fitness of 115.

The function consists of 4 *levels*, defined in Table 1. The first level consists of *elementary schemata,* each of which is a block of 8 1s. Each higher level consists of *compound schemata* composed of schemata from the previous one. The elementary and compound schemata are all beneficial schemata. An algorithm is said to *reach* a level when it finds an individual that is an instance of at least one significant schema from that level.

If p_1 consisted only of schemata s_0–s_{14}, it would be a RR function, similar to $R2$ [19]. The additional schemata s_{15}–s_{31} are potholes. The potholes s_{15} and s_{16} prevent a single-mutation hillclimber, such as the RMHC [7], that has reached the first-level schema s_0

from reaching the second-level schema s_8. This is because every sequence of single-bit mutations that leads from s_0 to s_8 would force the hillclimber to go downhill in fitness through one of these potholes (assuming neither of them is present to begin with), which it cannot do. Similarly, the potholes s_{17} and s_{18} prevent it from reaching s_8 if it has reached s_1. The remaining potholes block the path to the other second-level schemata.

4 EXPERIMENTS ON p_1

I first compared the SGA against a variety of hillclimbers on p_1, to verify that it was more difficult than the RR function $R2$ for hillclimbers such as RMHC.

4.1 SIMPLE GENETIC ALGORITHM (SGA)

The SGA used a population of 512 individuals. Two offspring were produced for each pair of parents, and the entire population was replaced in each new generation. Standard one-point crossover was used with a probability of 0.7 per mating pair. Point mutation was applied to each offspring with a probability of 0.005 per allele (mutations simply flipped the allele from 0 to 1 or vice-versa). Fitness-proportionate, or "roulette wheel", selection was used, with σ-truncation scaling [9]:

$$f' = \max\{\min[f - (\bar{f} - c\sigma), 1.5], 0\}. \tag{2}$$

Here f' is an individual's scaled fitness, f is its unscaled fitness, $\bar{f}$ is the population average unscaled fitness, σ is the standard deviation of unscaled fitness in the population, and the scaling constant $c = 2$. The maximum and minimum possible scaled fitnesses are 1.5 and 0, respectively. The scaled fitnesses were then used to select the parents of the next generation. If $\sigma < 0.0001$, the unscaled fitnesses were used instead. Scaling appears to be necessary for the SGA to do well on these functions.

These parameters were chosen rather arbitrarily, since the goal is to find a class of functions that the SGA can optimize easily, without being sensitive to the exact parameter settings.

4.2 HILLCLIMBERS

When comparing the SGA with hillclimbers, it is important to report results from a variety of hillclimbers. In these experiments, I used the next-ascent hillclimber (NAHC), steepest-ascent hillclimber (SAHC), and random-mutation hillclimber (RMHC) described by Forrest and Mitchell [7], and Jones's crossover hillclimber (XOHC) [16].

The XOHC used differs from Jones's in that it repeats if the maximum number of jumps is reached, until the maximum number of evaluations has been performed. Jones's original algorithm also quit if no fitness increase was found within 10000 steps; the one used here does not.

4.3 PERFORMANCE CRITERIA

In order for one algorithm to outperform another in this study, it should do so over a wide range of reasonable performance metrics. I use the number of function evaluations needed to reach the optimum as the primary performance metric, under the assumption that function evaluation dominates an optimizer's running time. When one algorithm reaches the optimum more often than the other, I use the number of times the optimum is reached as the primary performance metric. When neither algorithm reaches the optimum, I use the number of evaluations needed to reach each level, as well as the number of times each level was reached. The maximum fitness reached in each run is also recorded, but not shown here.

Fitness timeseries were also plotted for each algorithm, sampled every 512 function evaluations and averaged over the set of runs; only those for the SGA and two of the hillclimbers are shown here. (Individual runs were also plotted but are not shown here.) These provide much more information about the course of each run, including the rate of improvement in fitness. For the SGA, the population best and average unscaled fitness are plotted; for the hillclimbers the fitness of the best individual evaluated so far is plotted, along with the fitness of the current individual being evaluated.

4.4 EXPERIMENTAL RESULTS

The SGA, NAHC, SAHC, RMHC, and XOHC were each run 50 times on p_1, for 256000 function evaluations per run. The results are presented in Table 2. Timeseries for the SGA and RMHC are presented in Figures 1–2. The function p_1 achieved the goal of being hard for the RMHC. Neither it nor any of the other hillclimbers ever found the optimum. Among these algorithms, only the SGA ever found the optimum (level 4), and it did so in almost every run. The timeseries for the SGA and the RMHC are consistent with the other performance metrics.

5 LAX RANDOM-MUTATION HILLCLIMBER (LRMHC)

Following the dialectical heuristic presented in Section 2, the next step was to see how hard it was to design a hillclimber that outperformed the SGA on p_1. Since the fitness penalty of each pothole was 0.1, and the fitness contribution of each beneficial schema was

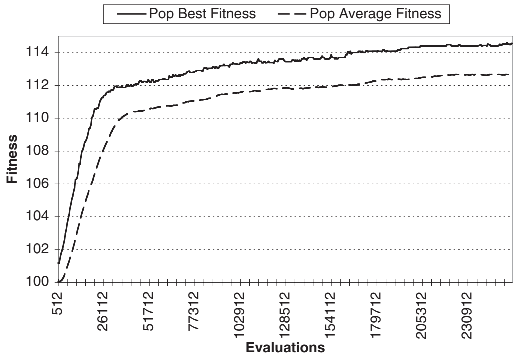


Figure 1: SGA population best and average fitness on p_1, sampled every generation and averaged over 50 runs.

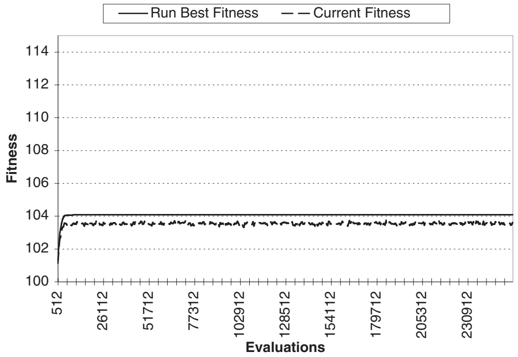


Figure 2: RMHC best and current fitness on p_1, sampled every 512 function evaluations and averaged over 50 runs.

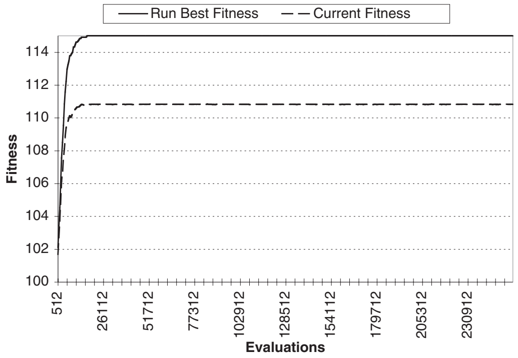


Figure 3: LRMHC best and current fitness on p_1, sampled every 512 function evaluations and averaged over 50 runs. ($\epsilon = 0.1$).

Table 2: The SGA, RMHC, NAHC, SAHC, XOHC, and LRMHC on p_1, 50 runs, 256000 function evaluations: Mean evaluations needed to reach each level (level 4 is the optimum). Here n is the number of times a level was reached, $\bar{x}$ is the sample mean number of evaluations needed to reach each level, averaged over n, and s is the sample standard deviation of the number of evaluations.

		Level			
		1	2	3	4
SGA	n	50	50	50	48
	$\bar{x}$	36.9	4026.9	17838.2	65063.2
	s	32.6	1833.5	13546.4	50361.2
NAHC	n	50	48	0	0
	$\bar{x}$	224.6	72733.2	—	—
	s	219.1	62128.0	—	—
SAHC	n	50	48	0	0
	$\bar{x}$	213.3	71687.3	—	—
	s	183.1	57099.5	—	—
RMHC	n	50	2	0	0
	$\bar{x}$	333.7	3133.5	—	—
	s	289.6	1051.5	—	—
XOHC	n	50	50	34	0
	$\bar{x}$	392.0	10454.8	110519.0	—
	s	453.8	8675.6	63414.0	—
LRMHC	n	50	50	50	50
($\epsilon = 0.1$)	$\bar{x}$	249.9	1342.5	3547.1	6244.0
	s	229.0	915.0	2025.4	3055.2

either 1.0 or 1.4, there is a foolproof method for determining whether a drop in fitness resulted from encountering a pothole, or from something else. If the drop is 0.1, then it is due to just a pothole and can be ignored. If it greater than or equal to 0.8, one or more beneficial schemata have been lost.

The lax random-mutation hillclimber (LRMHC) listed in Algorithm 2 resulted from incorporating this domain-specific knowledge into the RMHC. The LRMHC is exactly like the RMHC, except that it accepts any new string whose fitness is no more than ϵ below the fitness of the current string; in these experiments, ϵ is 0.1. (On p_1, ϵ can be any value in the interval $[0.1, 0.8)$.) The algorithm is very similar to the constant threshold algorithm (CTA) developed independently by Quick et al. [23]. The only difference is that the CTA accepts a new string only if its fitness is strictly greater than the old string's fitness minus ϵ, rather than greater than or equal as in the LRMHC. (Due to a typo, the LRMHC algorithm published by Holland [13] also differs in this way from the algorithm listed here.) In turn, both algorithms are similar to the record-to-record travel algorithm and the great deluge algorithm [3], and to threshold accepting [4].

1. Initialize the current individual to a random string.
2. Mutate one randomly-chosen allele. If the new string has a fitness equal to or greater than the current individual's fitness minus ϵ, replace the current individual with the new individual.
3. If the number of fitness evaluations performed so far is less than the maximum, go to Step 2. Otherwise, stop.

Algorithm 2: The lax random-mutation hillclimber (LRMHC) algorithm. ϵ is set to 0.1 in this paper.

In effect, the LRMHC assumes that any function it encounters has potholes that all have a depth of no more than ϵ, and that it can ignore them since building blocks have an observed fitness contribution higher than this value. It might seem unreasonable to incorporate this knowledge into the LRMHC. But the RMHC can be viewed as incorporating just as much knowledge; it merely assumes that the pothole depth is always 0. The issue here is not whether the LRMHC is a useful general-purpose optimizer for real functions. Rather, it is: How much domain-specific knowledge must be built into a hillclimber so that it outperforms the SGA on p_1?

5.1 LRMHC EXPERIMENTAL RESULTS

Results for LRMHC on p_1 are shown in Table 2 and Figure 2. It outperforms the SGA and all of the other algorithms on p_1, always finding the optimum, and finding it much faster than the SGA. The timeseries for LRMHC also shows a rapid increase in fitness over the SGA and RMHC. These results demonstrate that there is a very simple algorithm that outperforms the SGA by a wide margin on p_1. Similarly, Quick et al. [23] showed that the CTA outperformed a GA variant on a class of RR functions proposed by Holland and described by Jones [15], which also contained potholes. (However, the SGA outperforms this class of hillclimber on the RR function $R4$ [20].)

6 CONCLUSIONS

This paper has presented a research program to investigate the validity of the BBH, as well as some preliminary results. A dialectical heuristic for finding a simple class of functions on which the SGA outperformed other simple search algorithms was presented. A class of pothole functions was designed by adding potholes to the RR functions, in order to make them harder for simple hillclimbers. The pothole function p_1 was shown to be hard for hillclimbers such as the RMHC but easy for the SGA. Then a new hillclimber, the LRMHC, was designed by incorporating domain-specific knowledge about p_1 into the RMHC. While LRMHC is not useful as a general-purpose optimizer, this simple hillclimber outperformed the SGA on p_1, demonstrating that simple pothole functions such as p_1 are still too easy for hillclimbers. This result does not by itself invalidate the BBH. However, it reinforces the finding of the RR papers that simple assumptions about what functions are especially easy for the SGA, relative to other optimizers, are often unjustified. Simple hillclimbers can be surprisingly effective at optimizing simple functions.

The next step, following the dialectical heuristic from Section 2, is to modify p_1 so that it becomes hard for the LRMHC, while remaining easy for the SGA. First, however, the LRMHC's behavior on p_1 must be investigated, in order to predict what kinds of functions it will find difficult.

Acknowledgments

I thank the UM Royal Road Group, the UM Complex Systems Reading Group, and the anonymous reviewers for their comments and suggestions.

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, San Francisco, 1995. Morgan Kaufmann.

[2] K. A. De Jong. Genetic algorithms are NOT function optimizers. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 5–17, San Mateo, CA, USA, 1993. Morgan Kaufmann.

[3] G. Dueck. New optimization heuristics : The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104:86–92, 1993.

[4] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[5] R. A. Fisher. *The Genetical Theory of Natural Selection.* Dover, New York, second revised edition, 1958.

[6] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Learning.* IEEE Press, New York, 1995.

[7] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In

L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126, San Mateo, CA, USA, 1993. Morgan Kaufmann. Santa Fe Institute working paper 92-06-029.

[8] D. J. Futuyma. *Evolutionary Biology.* Sinauer, Sunderland, MA, USA, third edition, 1998.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison–Wesley, Reading, MA, USA, 1989.

[10] J. J. Grefenstette. Deception considered harmful. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75–91, San Mateo, CA, USA, 1993. Morgan Kaufmann.

[11] J. H. Holland. Re: Building block hypothesis considered harmful. *Genetic Algorithms Digest*, 5(20), 1991. URL `http://www.aic.nrl.navy.mil/galist/digests/v5n20`.

[12] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, USA, second edition, 1992. First edition: University of Michigan Press, Ann Arbor, 1975.

[13] J. H. Holland. Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation*, 8(4):373–391, 2000.

[14] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, MA, USA, 1986.

[15] T. Jones. A description of Holland's royal road function. *Evolutionary Computation*, 2(4):411–417, 1994.

[16] T. Jones. Crossover, macromutation, and population-based search. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80, San Francisco, 1995. Morgan Kaufmann.

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

[18] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1996.

[19] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In F. J. Varela and P. Bourgine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 245–254, Cambridge, MA, USA, 1992. MIT Press. Santa Fe Institute working paper 91-10-046.

[20] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, San Mateo, CA, USA, 1994. Morgan Kaufmann.

[21] A. Newell. *Unified Theories of Cognition.* Harvard University Press, Cambridge, MA, USA, 1990.

[22] U.-M. O'Reilly and F. Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, San Francisco, 1995. Morgan Kaufmann.

[23] R. J. Quick, V. J. Rayward-Smith, and G. D. Smith. The royal road functions: Description, intent and experimentation. In T. C. Fogarty, editor, *Evolutionary Computing: AISB Workshop, Brighton, U.K., April 1–2, 1996; Selected Papers*, volume 1143 of *Lecture Notes in Computer Science*, pages 223–235, Berlin, 1996. Springer.

[24] H.-P. Schwefel. *Evolution and Optimum Seeking.* Wiley, New York, 1995.

[25] J. A. Simpson and E. S. C. Weiner, editors. *The Oxford English Dictionary.* Oxford University Press, New York, second edition, 1989.

[26] G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, USA, 1989. Morgan Kaufmann.

[27] M. D. Vose. A critical examination of the schema theorem. Technical Report UT-CS-93-212, University of Tennessee Computer Science Department, Knoxville, TN, USA, 1993. URL `http://www.cs.utk.edu/~library/TechReports/1993/ut-cs-93-212.ps.Z`.

[28] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA, USA, 1999.

[29] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[30] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the Sixth International Congress of Genetics*, pages 356–366, 1932.

Crossing the Road to Efficient $\mathbb{ID}\mathbb{E}$As for Permutation Problems

Peter A.N. Bosman
Peter.Bosman@cs.uu.nl

Dirk Thierens
Dirk.Thierens@cs.uu.nl

Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

Abstract

In this paper, we introduce the $\mathbb{ICE}$ framework in which crossover from genetic algorithms (GAs) is incorporated in iterated density estimation evolutionary algorithms ($\mathbb{ID}\mathbb{E}$As). We focus on permutation optimization problems and show how pure continuous $\mathbb{ID}\mathbb{E}$As can be applied using the random keys representation. The problems that are hereby encountered, motivate the use of $\mathbb{ICE}$. As a result, permutation linkage information is effectively processed, resulting in efficient optimization of deceptive permutation problems of a bounded order. Experiments show that $\mathbb{ICE}$ outperforms pure continuous $\mathbb{ID}\mathbb{E}$As. Furthermore, we show that $\mathbb{ICE}$ gives insight into how new $\mathbb{ID}\mathbb{E}$As can be designed that efficiently work directly in the permutation search space.

1 Introduction

Finding and using the structure of the fitness landscape can aid evolutionary algorithms (EAs) in optimization. One approach to doing so, is by learning a probabilistic model from the selected samples and by using it in sampling new solutions. Such algorithms have been successfully applied to various problems in the case of discrete (binary) variables [9, 14, 15, 16, 17] and continuous (real) variables [4, 7, 13].

In this paper, we focus on the class of permutation optimization problems. This class of problems is very interesting. On the practical side, important real life problems such as scheduling and the traveling salesman are within this class. On the theoretical side, the search space grows *factorially* as the amount of decision variables increases. Furthermore, the solution space consists of permutations, which is fundamentally different from that of the binary or real space that most EAs have been designed for. EAs that directly work on permutations are usually equipped with specialized crossover operators that ensure feasibility of the solutions. An exception is the RKGA by Bean [1] in which the solutions are coded in the real space in such a way that all crossover operators generate feasible solutions. The RKGA has obtained good results especially in the field of scheduling.

With the exception of the OmeGA by Knazjew and Goldberg [12], no attempts have been made to learn and use the structure of permutation optimization problems. The OmeGA is essentially a *fast messy GA* [8] (fmGA) that has been adapted to work with the same coding as used in the RKGA. The OmeGA has been shown to effectively solve deceptive problems of bounded difficulty [11]. Furthermore, it significantly outperforms the RKGA, indicating the usefulness of finding and using problem structure.

The $\mathbb{ID}\mathbb{E}$A by Bosman and Thierens [4] is a framework for EAs in which probabilistic models are used. It has mostly been used to focus on continuous representations and optimization problems. Since the RKGA introduces a real coding of permutations, continuous $\mathbb{ID}\mathbb{E}$As can be directly applied. However, since the permutation space is discrete, such an application is likely to encounter problems. In this paper, we investigate the performance of continuous $\mathbb{ID}\mathbb{E}$As on permutation problems as a first attempt to design efficient probabilistic model building EAs for permutation problems. Furthermore, we explain the problems with such an approach and propose a solution for them by introducing crossover. The resulting algorithm shows efficient scaling behavior on problems of bounded complexity. The results indicate that $\mathbb{ID}\mathbb{E}$As can be designed that efficiently work directly in the space of permutations.

The remainder of this paper is organized as follows. In section 2 we discuss the $\mathbb{ID}\mathbb{E}$A framework. Next, we go over the coding that is used in the RKGA in section 3 and present the permutation problems that we test the algorithms on. Subsequently, we test the continuous IDEAs in section 4. In section 5, we propose to use crossover in the IDEA to overcome some of the problems of the continuous IDEA on permutation problems. We test the resulting EAs in section 6. In section 7 we reflect on the computational requirements of learning and using probabilistic models in evolutionary optimization. We discuss future research in section 8 and present our conclusions in section 9.

2 The IDEA framework

The IDEA framework is a general definition of *Iterated Density Estimation Evolutionary Algorithms* that has mostly been used to focus on continuous representations and optimization problems. Let $\boldsymbol{\mathcal{L}} = (0, 1, \ldots, l-1)$. The rationale behind the framework can be explained by assuming to have an l–dimensional optimization problem $C(y\langle\boldsymbol{\mathcal{L}}\rangle) = C((y_0, y_1, \ldots, y_{l-1}))$. Without loss of generality, we assume that we want to *minimize* $C(y\langle\boldsymbol{\mathcal{L}}\rangle)$. With every problem variable y_i, we associate a continuous random variable Y_i. Without any prior information on $C(y\langle\boldsymbol{\mathcal{L}}\rangle)$, we might as well assume a uniform distribution over $\boldsymbol{\mathcal{Y}} = (Y_0, Y_1, \ldots, Y_{l-1})$. Therefore, we generate an initial (population) vector of n samples at random. Now we let $P^{\theta}(\boldsymbol{\mathcal{Y}})$ be a probability distribution that is uniform over all vectors $y\langle\boldsymbol{\mathcal{L}}\rangle$ with $C(y\langle\boldsymbol{\mathcal{L}}\rangle) \leq \theta$. Sampling from $P^{\theta}(\boldsymbol{\mathcal{Y}})$ gives more samples that evaluate to a value below θ. Moreover, if we know $\theta^* = \min_{y\langle\boldsymbol{\mathcal{L}}\rangle}\{C(y\langle\boldsymbol{\mathcal{L}}\rangle)\}$, a single sample gives an optimal solution. To use this in an iterated algorithm, we select $\lfloor\tau n\rfloor$ 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 obtain an estimate $\hat{P}^{\theta_t}(\boldsymbol{\mathcal{Y}})$ as an approximation to the true distribution $P^{\theta_t}(\boldsymbol{\mathcal{Y}})$. New samples can then be drawn from $\hat{P}^{\theta_t}(\boldsymbol{\mathcal{Y}})$ and be used to replace some of the current samples. A formal definition and more details on the IDEA framework can be found elsewhere [4].

A special instance of the IDEA framework is obtained if selection is done by taking the best $\lfloor\tau n\rfloor$ samples, the amount of *new* samples is set to $n - \lfloor\tau n\rfloor$ and all of these new samples are used to replace the worst $n - \lfloor\tau n\rfloor$ samples. This results in the use of *elitism* such that the search for θ^* is conveyed through a monotonically decreasing series $\theta_0 \geq \theta_1 \geq \ldots \geq \theta_{t_{\text{end}}}$. We call the resulting algorithm a *monotonic* IDEA.

One of the most important parts in the IDEA is the estimation of the probability distribution of the selected samples. One way of achieving this, is by finding a factorized probability distribution, which is a product of probability density functions (pdfs).

If only multivariate joint pdfs are used, the factorization becomes a *marginal product model*. We also call this an *unconditional factorization*. For the factorization to be valid, each multivariate joint pdf must describe a unique set of variables. To this end, we define the *node vector* $\boldsymbol{\nu}$ to be a vector of mutually exclusive vectors that each hold the indices of the variables that are contained in a single multivariate joint pdf. The union of all of these mutually exclusive vectors is exactly equal to $\boldsymbol{\mathcal{L}}$. For example, a valid unconditional factorization for $l = 5$ is $\boldsymbol{\nu} = ((0, 4), (1), (3, 2))$.

If multivariate conditional pdfs are used, the product consists of exactly l factors. For each variable Y_i, we have exactly one factor $P(Y_i|Y\langle\pi(i)\rangle)$. The parent variables $Y\langle\pi(i)\rangle$ that Y_i is conditioned on, are indicated by a function $\pi(\cdot)$ that returns a vector $\pi(i) = (\pi(i)_0, \pi(i)_1, \ldots, \pi(i)_{|\pi(i)|-1})$. This can be identified with a directed graph by introducing a node i for every random variable Y_i. Furthermore, an arc $Y_i \to Y_j$ in the graph represents the fact that $i \in \pi(j)$. To be able to sample from the resulting factorized probability distribution, values must have been sampled for the parents of Y_i before sampling a value for Y_i itself. To ensure this, we use a vector of ordering variable indices $\boldsymbol{\omega} = \omega\langle\boldsymbol{\mathcal{L}}\rangle$. We can now enforce that while scanning the variables in the order $Y_{\omega_{l-1}}, Y_{\omega_{l-2}}, \ldots, Y_{\omega_0}$, the variables that some variable is conditioned on, will already have been regarded. This ordering can be found by performing a topological sort on the factorization graph. A valid conditional factorization for $l = 5$ is $\forall_i\,[\omega_i = i]\,, \pi(0) = (2), \pi(1) = (2, 3, 4), \pi(2) = (3, 4), \pi(3) = (4), \pi(4) = (0)$.

To find a good factorization, we use an incremental algorithm that starts from the univariate factorization in which each variable is independent from the others. For unconditional factorizations this means $\boldsymbol{\nu} = ((0), (1), \ldots, (l-1))$. For conditional factorizations we have $|\pi(i)| = 0, i \in \boldsymbol{\mathcal{L}}$. Each iteration, a single operation is performed on the factorization that increases some metric the most. In the unconditional case, the only operation we allow is merging two vectors. In the conditional case, we only allow the addition of a variable to the vector of parents of another variable. This corresponds to adding arcs to the factorization graph. However, to ensure that the conditional factorization is still valid, an arc is only allowed to be added if it does not introduce any cycles. If no operation that increases the metric can be performed anymore, the factorization search algorithm stops. In this paper, we use two metrics that should be minimized, which are the *Akaike Information Criterion* (AIC) and the *Bayesian Information Criterion* (BIC). For a derivation of these metrics in the ID$\mathbb{E}$A context, we refer the reader to a more detailed report [5]. The metrics are intended to provide a useful tradeoff between complexity and goodness of fit. Both metrics initially score a factorization by the negative log–likelihood of the factorized probability distribution. The AIC metric penalizes complexity by adding the amount of parameters $|\boldsymbol{\theta}|$ that has to be fit. Note that the operations on factorizations that we have proposed, always increase the amount of parameters. The BIC metric is parameterized by a regularization parameter λ that determines the amount of penalization of more complex models in order to favor more simple models. The penalization in the BIC metric increases logarithmically with the size of the sample vector $\lambda \ln(|\boldsymbol{\mathcal{S}}|)|\boldsymbol{\theta}|$. In this paper, we have used $\lambda = \frac{1}{2}$. As the size of the sample vector increases, the BIC metric has a stronger penalization than does the AIC metric. Since we have fixed λ to $\frac{1}{2}$, this is the case for any practical population size.

3 Random keys and permutation problems

Permutations can be encoded in the real space using random keys. In this section, we briefly go over this encoding and present some permutation problems of tunable bounded difficulty.

3.1 Permutation encoding

The encoding of permutations by random keys was introduced by Bean [1]. The main advantage of random keys is that no crossover operator can create unfeasible solutions since each encoding represents a permutation. To encode a permutation of length l, each integer in $\boldsymbol{\mathcal{L}}$ is assigned a value (key) from some real domain, which is usually taken to be $[0, 1]$. Subsequently, the numbers in $\boldsymbol{\mathcal{L}}$ are sorted on the keys to get the resulting permutation. For example, the random key string $(0.25, 0.1, 0.9)$ represents the ordering $(1, 0, 2)$. This decoding requires $\mathcal{O}(l \log l)$ time.

3.2 Problems of bounded order

Knazjew [11] has introduced a general deceptive permutation problem. The advantage of this problem over most test problems is that the interactions between the random keys are restricted to be of a certain order l_{BB}. A further advantage is that it is defined for any l_{BB}, unlike for instance the deceptive permutation problems by Kargupta, Deb and Goldberg [10].

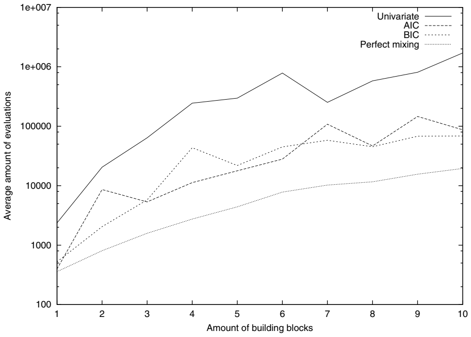


Figure 1: Amount of evaluations for the continuous ID$\mathbb{E}$A, $l_{BB} = 4$.

Knazjew's deceptive permutation problem is defined using a distance between two permutations. This distance is defined as the minimum number of elements in one string to be moved to obtain the other string. Furthermore, the optimum is the trivial permutation $(0, 1, \ldots, l_{BB} - 1)$. The distance from any permutation $\boldsymbol{y}$ to the optimum equals $l_{BB} - |\text{LIS}(\boldsymbol{y})|$, where $\text{LIS}(\boldsymbol{y})$ is the longest increasing subsequence in $\boldsymbol{y}$. For example, if $\boldsymbol{y} = (4, 0, 3, 1, 2)$, then $\text{LIS}(\boldsymbol{y}) = (0, 1, 2)$ and $l_{BB} - |\text{LIS}(\boldsymbol{y})| = 5 - 3 = 2$. The two elements to move are of course 4 and 3. Note that the reverse permutation $(l_{BB} - 1, l_{BB} - 2, \ldots, 0)$ is the only permutation with a distance of $l_{BB} - 1$. The deceptive ordering problem for a single *building block* (BB) of length l_{BB}, is defined as follows:

$$f_{BB}(\boldsymbol{y}) = \begin{cases} 1 - \frac{|\text{LIS}(\boldsymbol{y})|}{l_{BB}} & \text{if } |\text{LIS}(\boldsymbol{y})| < l_{BB} \\ 1 & \text{if } |\text{LIS}(\boldsymbol{y})| = l_{BB} \end{cases} \qquad (1)$$

A building block is a subsequence of the complete random key string. The actual fitness function that we use, has length $l = n_{BB} l_{BB}$, where n_{BB} is the amount of building blocks. The locations of the individual building blocks have been coded *loosely*. This implies that building block $0 \leq i < n_{BB}$ consists of the random keys found at locations $(i, i + n_{BB}, \ldots, i + (l_{BB} - 1)n_{BB})$. The fact that the problem is fully deceptive, means that all schemata of an order smaller than k lead to the suboptimum of the reverse permutation [10]. This makes the problem hard for any optimizer that doesn't identify which random key positions together constitute a building block [3]. Furthermore, because of the loose coding, simple crossover schemes such as one point crossover are not effective either.

4 ID$\mathbb{E}$As based on normal pdfs for permutation problems

Since random key strings are real numbers, we can directly apply continuous ID$\mathbb{E}$As to permutation prob-

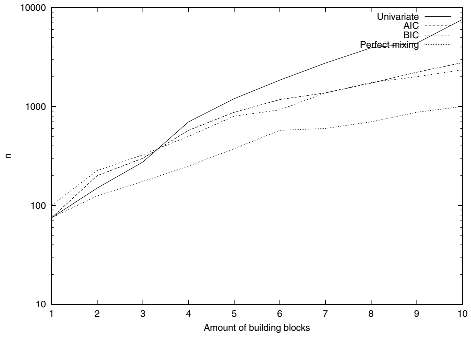


Figure 2: Required population size for the continuous ID𝔼A, $l_{BB} = 4$.

lems. In this paper, we use the normal pdf for the pdfs implied by the factorization. We refer the reader to previous work [4] for implementation details.

In all our testing, we used monotonic ID𝔼As. We used the rule of thumb by Mühlenbein and Mahnig [14] for FDA and set τ to 0.3. All results were averaged over 30 independent runs. As a measure of efficiency, we use the average amount of required function evaluations.

In figures 1 and 2, the average amount of evaluations and the required population size are shown respectively on a logarithmic scale for $n_{BB} \in \{1, 2, \ldots 10\}$ and $l_{BB} = 4$. It is clear that using the univariate factorization in which each variable is taken independent of all of the others, scales up significantly worse than when problem structure is exploited. The structure of the problem is best represented in an unconditional factorization in which the building blocks are perfectly separated in the node vector $\boldsymbol{\nu}$. We call this *perfect mixing* information. Using this structure can be seen to scale up polynomially. There is no obvious difference between the AIC or BIC search metric here.

Note that the amount of instances for a permutation of $l_{BB} = 4$ is $4! = 24$, which still tractable. If we move to $l_{BB} = 5$ however, the amount of instances already becomes 120, which significantly increases the problem difficulty. This is reflected by the fact that the continuous ID𝔼As are unable to solve the deceptive problems for $n_{BB} \in \{3, 4, \ldots 10\}$ with $n \leq 1.0 \cdot 10^5$. In the case of perfect mixing information, we got 2.0 BBs on average using $3.1 \cdot 10^6$ evaluations and $n = 3.0 \cdot 10^4$. The OmeGA significantly outperforms the continuous ID𝔼A, which points out that the continuous ID𝔼A approach less efficiently processes the building blocks.

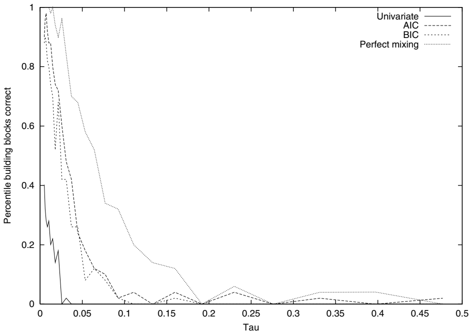


Figure 3: Optimization performance of the continuous ID𝔼A for increasing τ, $l_{BB} = n_{BB} = 5$, $\lfloor \tau n \rfloor = 250$.

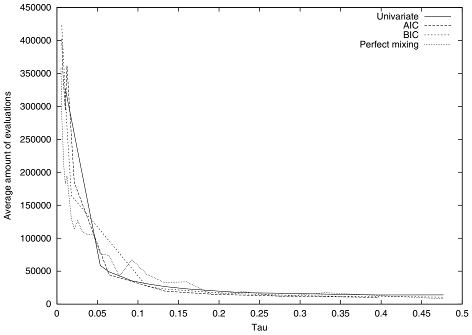


Figure 4: Function evaluations of the continuous ID𝔼A for increasing τ, $l_{BB} = n_{BB} = 5$, $\lfloor \tau n \rfloor = 250$.

5 The 𝕀C𝔼 framework

The results of the continuous ID𝔼A cannot compete for instance with the OmeGA. Both algorithms however attempt to find and use the relations between the random keys. The main problem with the continuous ID𝔼A is the density estimation of and sampling from a normal pdf. It has some limitations with respect to non–linearity and multimodality. Furthermore, the permutation space is inbedded in the real space, meaning that the actual search space is a (special) discretization of $[0, 1]^l$. For an l_{BB}–dimensional f_{BB} function, the optimum and the suboptimum are contained in a convex region consisting of $\frac{100}{l_{BB}!}\%$ of the $[0, 1]^{l_{BB}}$ hypercube. Furthermore, these two regions are separated by the single line on which every random key has an identical value. Finally, the other types of building blocks have the highest fitness in the neighborhood of the suboptimal block. This means that, especially as l_{BB} goes up, the geometrical approximation of the normal distribution will have great difficulty to represent for instance the trade–off between the suboptimal block and the optimal block. To increase their separability, the selection pressure should increase or the population size should increase. For instance, in order to isolate the optimal building block in a random population, a selection percentage of $\tau = \frac{100}{l_{BB}!}\%$ is required. The performance for different values of τ is displayed in figures 3 and 4. In these figures, the selection size $\lfloor \tau n \rfloor$ is kept constant. The rationale for this is that the goodness of a normal pdf fit does no longer significantly increase if the amount of samples increases. The optimization performance increases if the selection pressure increases. However, this comes at the expense of the amount of required evaluations. Therefore, there must be some range for the best choice for τ in the sense of required amount of evaluations to reach the optimum. In section 7, we investigate this further.

To overcome the problems with the normal pdf in the real space with respect to fitting the embedded permutation space, the random key strings should be interpreted in the permutation space. By sampling from the continuous normal pdf, we introduce a lot of redundancy, since the random key string $(0.1, 0.2, 0.3)$ codes the same permutation as $(0.91, 0.99, 0.999)$. To cope with this redundancy, instead of sampling *new* building blocks from a normal pdf, we propose to *mix* them using crossover. Only combinations of the initial strings are thereby generated, just as is done in GAs.

The way in which crossover is done, determines the rate of success. First, we note that *if* the building blocks are exchanged between parents as a whole, the information is mixed in the most efficient mixing manner, resolving the redundancy problem. However, on beforehand we generally don't know the location and size of the building blocks. This information is called *linkage information*. To find this information, we rely on the remainder of the ID𝔼A framework to find a structure that contains this linkage information. For instance, if we have an unconditional factorization, we are given groups of random keys that should be processed in a multivariate joint pdf. Therefore, these random keys should be processed together as a block. We hope that these blocks are a good approximation of the true building blocks in the problem.

Before copying a block to the offspring, each block may be transformed using a function $\varrho(\cdot)$. This function rescales the random keys to a subinterval of $[0, 1]$ with probability p_ϱ. If we for instance scale $(0.1, 0.2, 0.3)$ to $[0.9, 0.95]$, we get $(0.9, 0.925, 0.95)$. Note that this doesn't change the permutation that is encoded. The optimization problem can require a relative ordering of the building blocks. Without rescaling, we have to rely on the random key combinations that are generated initially. Rescaling the blocks increases the probability that they will be combined properly. To ensure a large enough amount of intervals so that the blocks themselves can be ordered, we set this amount to l.

We call the framework for the resulting algorithms 𝕀C𝔼 (ID𝔼A *Induced Chromosome Elements Exchanger*). Its definition equals that of the ID𝔼A, with the exception of the creation of offspring. In 𝕀C𝔼, this is not done by sampling, but by randomly selecting two parents from the selected samples and crossing over blocks in the solutions. Which blocks we actually perform crossover with, is determined by the type of factorization. Since the resulting algorithm uses crossover, it can validly be argued that we have designed a GA. The specialty of this GA is that it attempts to learn linkage information for permutations and use this linkage information in a linkage preserving crossover operator.

```
𝕀C𝔼
$par_0 \leftarrow$ RANDOM($\{0, 1, \ldots, \lfloor \tau n \rfloor - 1\}$)
$par_1 \leftarrow$ RANDOM($\{0, 1, \ldots, \lfloor \tau n \rfloor - 1\} - \{par_0\}$)
$\boldsymbol{\mathcal{B}} \leftarrow$ CROSSOVERBLOCKS($\varsigma$)
for $i \leftarrow 0$ to $|\boldsymbol{\mathcal{B}}| - 1$ do
    $par \leftarrow$ RANDOM($\{par_0, par_1\}$)
    for $j \leftarrow 0$ to $|\boldsymbol{\mathcal{B}}_i| - 1$ do
        $\boldsymbol{off}_{(\boldsymbol{\mathcal{B}}_i)_j} \leftarrow \varrho((\boldsymbol{y}^{\boldsymbol{par}})_{(\boldsymbol{\mathcal{B}}_i)_j})$
RETURN($\boldsymbol{off}$)
```

6 Specific crossover operators in 𝕀C𝔼

Having introduced the 𝕀C𝔼 framework, we test it on the deceptive permutation functions. First however, we elaborate on how we have selected the blocks to be crossed over in our experiments.

6.1 Crossover operators

Conditional factorizations allow for more precise probabilistic modelling because unconditional factorizations can be expressed using conditional factorizations, but not vice versa. Therefore, it is interesting to base crossover operators on both types of factorizations.

6.1.1 Unconditional: block mixing

By crossing over the random keys based on the node vector $\boldsymbol{\nu}$, we attempt to directly exchange and mix the important blocks of information. This results in an approach similar to the ECGA by Harik [9].

```
CROSSOVERBLOCKS($\boldsymbol{\nu}$)
RETURN($\boldsymbol{\nu}$)
```

6.1.2 Conditional: position biased TPX

To use conditional factorizations, we learn a chain of dependencies in which random keys that are important together, are placed close to each other. Subsequently, we can apply for instance two point crossover such that the linkage information in the chain is respected. This approach was recognized earlier [3] to be interesting for processing linkage information. To find a chain,

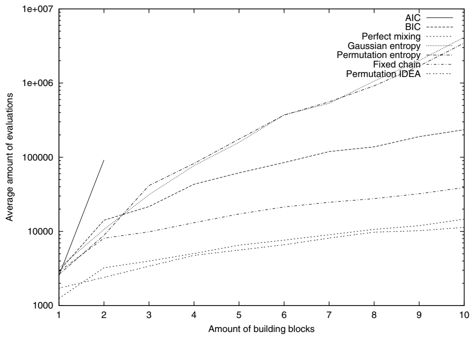


Figure 5: Amount of evaluations for $\mathbb{ICE}$, $l_{BB} = 5$.

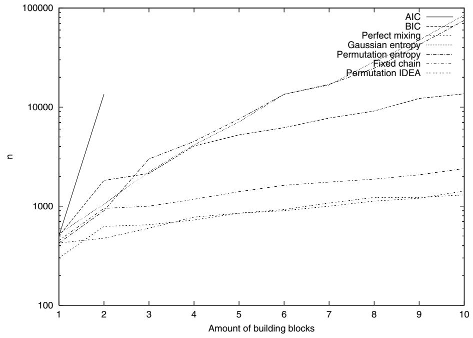


Figure 6: Required population size for $\mathbb{ICE}$, $l_{BB} = 5$.

the greedy entropy algorithm in MIMIC [2] can be used. The entropy, which equals the average negative log–likelihood for normal pdfs [6], can be computed using the normal pdf as we have done for all continuous ID$\mathbb{E}$As, but we can also use permutation entropy. This amounts each time to finding the unselected node Y_{i+1} that occurs more often in one ordering than in the other with respect to the lastly selected node Y_i (maximize $|\hat{P}(Y_i < Y_{i+1}) - \hat{P}(Y_{i+1} < Y_i)|$). The resulting blocks to be exchanged, can now be found as follows:

```
CROSSOVERBLOCKS($(\pi, \boldsymbol{\omega})$)
$pos_0 \leftarrow$ RANDOM($\{0, 1, \ldots, l\}$)
$pos_1 \leftarrow$ RANDOM($\{0, 1, \ldots, l\}$)
if $pos_1 > pos_0$ then
    $pos_0 \leftrightarrow pos_1$
RETURN($((\omega_0, \omega_1, \ldots, \omega_{(pos_0-1)})$,
        $(\omega_{pos_0}, \omega_{(pos_0+1)}, \ldots, \omega_{(pos_1-1)})$,
        $(\omega_{pos_1}, \omega_{(pos_1+1)}, \ldots, \omega_{l-1}))$)
```

6.2 Results

$\mathbb{ICE}$ gives significantly better results (figures 5 and 6). All tested algorithms, with the exception of unconditional factorizations in combination with the AIC metric, were able to solve all tested problems up to $n_{BB} = 10$ for $l_{BB} = 5$, with population sizes (well) below $1 \cdot 10^5$. The entry of the permutation ID$\mathbb{E}$A is explained in section 8. Because the deceptive problem is better separable if the selection pressure increases, it was again observed that the performance increases if τ decreases for a fixed $\lfloor \tau n \rfloor$ in a similar fashion as observed for continuous ID$\mathbb{E}$As (figures 3 and 4).

Since the AIC metric did not penalize the likelihood effectively, it resulted in unconditional factorizations that combined entire building blocks to sometimes form the complete joint factorization, which does not lead to efficient exploration of the permutation space. In this case there is thus a preference for the BIC metric. All approaches can be seen to scale up polynomially. However, the chain approaches that use search metrics scale up significantly worse. There is not much to choose between the normal entropy or the permutation entropy. If the right structure is found (fixed chain), the building blocks are propagated effectively using position biased TPX. However, it is hard to find the right second order linkage information, as these approaches clearly scale up a lot worse for increasing n_{BB}. We conclude in a similar fashion as has been done for binary spaces [3], that finding and using lower order linkage information is less efficient than finding and using higher order linkage information.

From the results shown so far, it seems that the best approach is to use an unconditional factorization with the BIC information criterion. However, we have not tested it yet on problems with conditional dependencies. To test this, along with the influence of random rescaling, we have used a difficult permutation problem with overlapping building blocks. All of the building blocks are now coded sequentially instead of loosely. For $n_{BB} = 3$ and $l_{BB} = 4$, the individual building blocks are found at positions $(0, 1, 2, 3)$, $(3, 4, 5, 6)$ and $(6, 7, 8, 9)$, giving $l = 10$. Note that the sole optimal solution is the complete trivial permutation of length l. We tested the algorithms on a problem with $l = 40$, giving $n_{BB} = 13$ at $l_{BB} = 4$. We tested population sizes up to $n = 10^5$ and allowed for a maximum of 10^9 evaluations. The results are shown in figure 7. It becomes clear that using conditional dependencies in a chain can be efficient, especially when random rescaling is used. However, the results also show again that learning the chain to use for position biased TPX does not lead to the best results. Furthermore, introducing random rescaling *does* improve the results of the unconditional factorization. The best reported results of the OmeGA are approximately $\overline{BBs} = 9.60$ after $300 \cdot 10^6$ evaluations.

7 A note on the running times

Algorithms that build and use probabilistic models take up more time every iteration as the complexity

Unconditional				
$\mathfrak{f}$	$\boldsymbol{p_\varrho}$	$\boldsymbol{n \cdot 10^4}$	$\overline{\boldsymbol{BBs}}$	$\overline{\boldsymbol{Eval}} \cdot \mathbf{10^6}$
BIC	0	7.5	8.23	2.0
BIC	0.5	10.0	10.61	481.1

Conditional				
$\mathfrak{f}$	$\boldsymbol{p_\varrho}$	$\boldsymbol{n \cdot 10^4}$	$\overline{\boldsymbol{BBs}}$	$\overline{\boldsymbol{Eval}} \cdot \mathbf{10^6}$
NE	0	9.5	7.60	3.2
NE	0.5	4.5	7.23	3.9
FIX	0	8.5	11.17	2.9
FIX	0.5	0.1	13.00	0.023

Figure 7: Results on the overlapping deceptive permutation problem, $l_{BB} = 4$, $n_{BB} = 13$, ($l = 40$, NE = normal pdf entropy, FIX = fixed optimal chain).

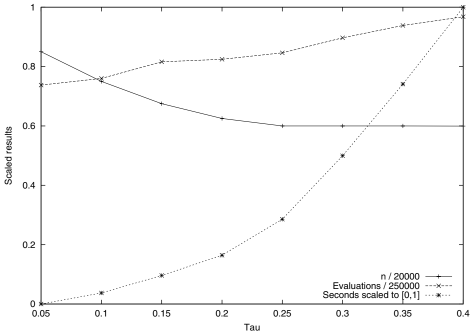


Figure 8: $\mathbb{ICE}$, $l_{BB} = 5$, $n_{BB} = 10$, 100% successrate.

of the models increases. It is therefore important to be aware of the running time next to the amount of function evaluations. If we have a very costly evaluation function, this is of less importance. However, there are plenty practical optimization problems, such as knapsack and traveling salesman, that can be evaluated efficiently and easily result in a very large dimensionality l. Since the learning of higher order factorizations often scales as $\mathcal{O}(l^3)$, exploiting problem structure by learning factorizations will not result in fast algorithms as l increases.

In this paper, we have used the rule of thumb by Mühlenbein and Mahnig [14] for FDA. However, this rule of thumb is based on observations on non–deceptive binary problems. Furthermore, the elitism in FDA is restricted to the single best solution of the previous generation. In figure 8, we have plotted the most efficient result of $\mathbb{ICE}$ using the BIC search metric on unconditional factorizations for different values of τ when each of 30 independent runs reached all building blocks correct. When only the amount of evaluations or the actual complete running time is important, $\tau = 0.3$ is clearly not optimal. Since the separability of the deceptive problem becomes easier when a greater selection pressure is applied, this is not surprising. For the best memory–computation time trade–off, $\tau \in [0.15, 0.25]$ seems to be an effective choice. Concluding, we suggest from empirical observervations to use $\tau \leq 0.25$ in future experiments.

8 Discussion and future research

In this paper, we have used penalization metrics to guide the search for a good factorization. The difficulty with such an approach is choosing the right amount of regularization. The amount of regularization can be seen as a parameter that defines how much time the algorithm is allowed to spend on model building. In a way, this is a substitute for limiting the maximum order of interactions in the factorizations. However, using a metric in addition influences the decision of which operations on the factorization are beneficial. Seen in this way, a penalization metric is a practical approach to using exact statistical hypothesis tests to determine whether some operation is truly beneficial. Even though the use of statistical hypothesis tests is exact, it is also of less practical use because of its computational requirements [6]. Therefore, it is practically more interesting to use a metric as an approximation. Still, it should be noted that selecting the settings for such a metric is always subject to user experience.

We note that in this paper we have only run tests over a limited amount of problems. Even though the results are encouraging, verification on other problems is desired. It would for instance be interesting to see how $\mathbb{ICE}$ algorithms perform on real life scheduling problems as opposed to other EA approaches.

The $\mathbb{ICE}$ framework has pointed out that effective EAs for permutation problems can be constructed if we find and use linkage information in the permutation space directly. Since crossover on permutations is used in $\mathbb{ICE}$, this is evidence that efficient $\mathbb{IDEA}$s can be designed by making them process the permutation space directly. To do so, we need a way to estimate the multivariate joint distribution of a set of random keys. This is sufficient to also process conditional factorizations, since a conditional probability is a quotient of two multivariate joint probabilities. In order to estimate the probability of a set of k random keys, we can count the frequency of a certain permutation instance. This means that we require a frequency table of minimum size $k!$. To generate such a table of minimum size, we need to map permutations onto integers. This can be done in $\mathcal{O}(k \log k)$ time whereas a new random key sequence can be made from an integer in $\mathcal{O}(k)$ time. Given this correspondence, the frequencies of the possible permutations can be counted from the selected samples. Subsequently, new samples can be drawn in a similar fashion as is done for binary variables.

In figures 5 and 6, the results of testing a permutation based ID𝔼A with the best possible fixed unconditional factorization, are plotted. At this point, the development of the permutation based ID𝔼A only allows fixed factorizations as input as is the case for the FDA [14]. However, the results in this section indicate that ID𝔼As may be constructed by learning factorizations and thereby no longer assuming a priori knowledge on the optimization problem. The results of the permutation ID𝔼A are slightly better than those of 𝕀ℂ𝔼 with perfect mixing information. However, we have provided the permutation ID𝔼A with the perfect a priori structure information. It would be interesting to see if a permutation ID𝔼A that learns this structure can outperform 𝕀ℂ𝔼 on various problems.

9 Conclusions

Finding and using problem structure can aid EAs. We have shown that this is also the case when random keys are used to tackle permutation problems. Furthermore, finding and using a structure in which a multiple of variables interact, is superior to using only second order linkage information when the optimization problem contains higher order interactions.

We have shown by introducing 𝕀ℂ𝔼 that by mixing blocks of random keys, permutation problems of a bounded difficulty can be solved efficiently. To this end, we have used continuous ID𝔼As to find the structure of the optimization problem together with crossover from GAs to mix the building blocks. By doing so, the EA directly explores the permutation space, which significantly increases its efficiency. This is an indication that ID𝔼As that are designed to work directly in the space of permutations may effectively find and use the structure of permutation problems.

References

[1] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994

[2] J.S. De Bonet, C. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing*, 9, 1996

[3] P.A.N. Bosman and D. Thierens. Linkage information processing in distribution estimation algorithms. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proc. of the GECCO–1999 Genetic and Evol. Comp. Conference*, pages 60–67. M.K. Pub., 1999

[4] 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, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 767–776. Springer, 2000

[5] P.A.N. Bosman and D. Thierens. Mixed ID𝔼As. Utr. Univ. Tech. R. UU–CS–2000–45. ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-2000/2000-45.ps.gz, 2000

[6] P.A.N. Bosman and D. Thierens. Negative log–likelihood and statistical hypothesis testing as the basis of model selection in ID𝔼As. In A. Feelders, editor, *Proceedings of the Tenth Dutch–Netherlands Conference on Machine Learning.* Tilburg University, 2000

[7] M. Gallagher, M. Fream, and T. Downs. Real–valued evolutionary optimization using a flexible probability density estimator. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 840–846. Morgan Kaufmann Publishers, 1999

[8] D.E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64. M.K. Pub., 1993

[9] G. Harik. Linkage learning via probabilistic modeling in the ECGA. IlliGAL Tech. R. 99010. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/99010.ps.Z, 1999

[10] H. Kargupta, K. Deb, and D.E. Goldberg. Ordering genetic algorithms and deception. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature – PPSN II*, pages 47–56. Springer, 1992

[11] D. Knazjew. Application of the fast messy genetic algorithm to permutation and scheduling problems. IlliGAL Technical Report 2000022. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/2000022.ps.Z, 2000

[12] D. Knazjew and D.E. Goldberg. Large–scale permutation optimization with the ordering messy genetic algorithm. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 631–640. Springer, 2000

[13] P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña. Optimization by learning and simulation of bayesian and gaussian networks. Univ. of the Basque Country Tech. R. EHU-KZAA-IK-4/99. http://www.sc.ehu.es/ccwbayes/postscript/kzaa-ik-04-99.ps, 1999

[14] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evol. Comp.*, 7:353–376, 1999

[15] A. Ochoa, H. Mühlenbein, and M. Soto. A factorized distribution algorithm using single connected bayesian networks. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 787–796. Springer, 2000

[16] M. Pelikan, D.E. Goldberg, and E. Cantú-Paz. BOA: The bayesian optimization algorithm. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 525–532. M.K. Pub., 1999

[17] M. Pelikan, D.E. Goldberg, and K. Sastry. Bayesian optimization algorithm, decision graphs and occam's razor. IlliGAL Tech. R. 2000020. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/2000020.ps.Z, 2000.

Agent Motion Planning with GAs Enhanced by Memory Models

Martijn C.J. Bot
Vrije Universiteit
Faculty of Science
De Boelelaan 1081
1081 HV Amsterdam
+31 20-4447790
mbot@cs.vu.nl

Neil Urquhart
School Of Computing
Napier University
219 Colinton Road
Edinburgh
+44 0131 455 4432
n.urquhart@napier.ac.uk

Ken Chisholm
School Of Computing
Napier University
219 Colinton Road
Edinburgh
+44 0131 455 4216
k.chisholm@dcs.napier.ac.uk

Abstract

The Tartarus problem may be considered a benchmark problem in the field of robotics. A robotic agent is required to move a number of blocks to the edge of an environment. The location of the blocks and position of the robot is unknown initially. The authors present a framework that allows the agent to learn about its environment and plan ahead using a GA to solve the problem. The authors prove that the GA based method provides the best published result on the Tartarus problem. An exhaustive search is used within the framework as a comparison, this provides a higher score still. This paper presents the two best Tartarus results yet published.

1 Introduction

The Tartarus problem may be considered a benchmark problem in the area of non-Markovian agent motion planning. The agent is placed within an environment, with no prior knowledge of the environment and limited means by which to gather information on the environment (see Figure 1). The task to be undertaken involves moving blocks placed at random positions within the environment to the outer edges of the environment. There is only a finite amount of energy available to the agent, thus limiting the number of moves that can be made.

The challenge is therefore to devise a solution to the problem that can gather information on the environment and solve the problem at the same time. We enhance a genetic algorithm with a long term memory model for incorporating information that was found in previous steps. We will show that our approach outperforms leading algorithms on this problem.

2 Problem Description

2.1 An overview of the Tartarus Problem

Within the Tartarus problem, a robotic agent is placed in an environment that consists of a 6x6 square grid (akin to a checkers board, see Figure 1). The agent occupies one square, while also on the board are 6 blocks each of which occupy one square. The object of the exercise is for the agent to push the blocks to the edge of the board, scoring 1 point for each block moved to an edge or two points for each block pushed into a corner. The maximum score then is 10. Only one block may be pushed at one time. Each time the agent moves forward, rotates or pushes a block forward it uses one unit of energy.

The agent's sensors can only detect the contents of the 8 squares directly surrounding the agent's position.

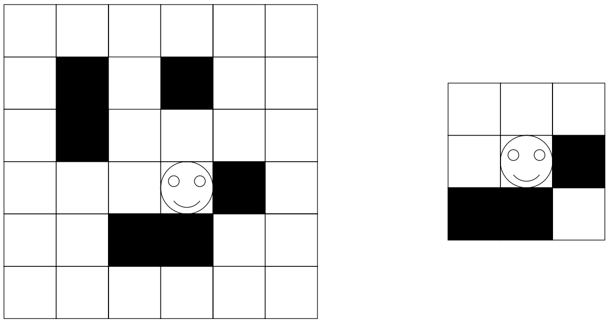
The objective of the agent is to maximize the average score over 100 randomly generated boards.

2.2 Board Initialisation

The board is initialised by placing all 6 blocks in random squares, and then placing the agent in a random square facing a random direction. Neither the blocks nor the agent will be initially placed adjacent to the edge. A configuration of 4 blocks placed together cannot be moved by the agent (because it can only move one block at a time). Therefore the board is never initialised with four blocks arranged in a square.

2.3 Sensors

The agents' sensors are capable of sensing the contents of the eight squares adjacent to the agents' current position. The sensors can detect whether each square is empty, contains a block, or constitutes part of an edge. The agent cannot sense its orientation or its



Tartarus board View of the agent

Figure 1: Example Tartarus board

position on the grid.

2.4 Energy Levels

Within the Tartarus problem, there is no time limit, but the agent has only a limited amount of energy. The agent has an initial energy level of 80 units, each move forward or rotation costs the agent 1 unit of energy. Once all the energy has been used, the agent can no longer move and the attempt to solve the problem ceases.

3 Previous Work

Previous techniques applied to the Tartarus Problem include genetic algorithms, neural networks, finite state machines and genetic programming.

Teller[Teller, 1994] used genetic programming with indexed memory to achieve an average score of 4.5.
In [Balakrishnan and Honavar, 1996], neural networks have been utilised with a maximum score of only 4.5.
The highest score achieved so far has been by [Ashlock and Joenks, 1998], whose GP-based algorithm averaged a score of 8.2. Earlier GP based work by Ashlock and McRoberts[Ashlock and McRoberts, 1997] achieved a score of 8.15.
The most recent research has been undertaken by [Ashlock and Freeman, 2000] who utilised a GA to evolve a finite state machine. The finite state machine interprets the results of the sensors and at each change in state can issue a command to the agent. The average final score achieved by Ashlock and Freeman was 7.11.
All of the research outlined above utilised some form of internal state or memory within the agent to allow the agent to learn about the environment. Solutions that haven't utilised some form of internal state within the agent have not achieved an average score of greater than 2.
By examining previous research, it may be concluded that the agent needs to be equipped with the ability to hold an internal state within some form of 'memory'.

4 Formulating the Solution

4.1 Human Attempts to Solve the Problem

The authors initially carried out an informal experiment using human agents (i.e. a human controlling the agent by manually issuing commands). One agent was asked to solve the problem while only being allowed to view the inputs from the eight sensors. The second agent was allowed paper and pencil to draw a map of the environment as they explored it. Each agent attempted to solve the problem 10 times. The experiment revealed that even with the processing power of a human brain, the efficiency of the solutions increased dramatically when the agent was allowed to collate the information gathered through its sensors in the form of a map. Without a map, the human agent averaged a score of 7.2, but with the energy levels reduced to 0 in every case. By allowing the human agent to build a map, the average score rises to 9.1 with more energy left.

The authors' perceived reason for the human agents improved performance when drawing a map, was the ability to use the information in the map to pre-plan sequences of moves before issuing commands to the agent. Cognitive psychologists have estimated human short-term memory only to capable of containing 7±2 'chunks' of information. The human agent working without the map may have been unable to recall the previous values of the sensors, and build a 'memory map' of the area.

4.2 A Description of the Chosen Solution

4.2.1 Overview

The information contained in the agents' sensors may be considered equivalent to the human short-term memory. They are both transient and of low capacity. The informal experiment conducted in section 4.1 and previous research reviewed in section 2 both suggested a requirement for the agent to be given some form of 'long-term' memory. This long-term memory will contain information about the environment, gathered from the short-term memory (sensors) as the agent is moved.

Having established the requirement for short and long-term memories, we now require to process the information stored in the long-term memory to allow the agent to carry out its task. The processor function will be carried out by a Genetic Algorithm (GA). The GA will evolve command sequences consisting of Forward, Left or Right moves to allow the agent to push the blocks discovered so far to the edge of the board. After a set number of evaluations the GA will be halted and the command sequence contained within the best chromosome will be executed by the agent. As soon as the agent discovers a new feature within the landscape, it stops executing the command sequence and the GA is restarted to evolve a new command sequence based on the updated information now contained within the long-term memory.

4.2.2 The Long and Short Term Memories

As has already been described, the short-term memory is the buffer for the eight sensors. Each time the agent moves, the information contained within the sensors will be replaced by values relating to the agents' new position.

The long-term memory is a 11x11 grid. The long-term memory must be bigger than the board, because the agent could initially be placed almost anywhere on the board. The long-term memory is large enough to allow the data sensed from the agents initial position to be placed in the centre and then the map to be built out from this point.

Each of the 121 locations within long-term memory can hold one of five values;

1. Block: This square definitely contains a block
2. Empty: This square is definitely empty
3. Edge: This square is on the edge
4. Probably Empty: This square has not been explored yet, but it is assumed that it is empty
5. Something: The agent has tried to push a block into this square, but couldn't as it is either occupied by another block or it forms part of the edge

As the agent progresses in solving the Tartarus problem, the map contained within long-term memory is built-up. This map is used by the GA fitness function (see section 4.2.4) when evaluating command sequences.

4.2.3 Wall Deduction Heuristics

Because the characteristics of the environment, its size, shape and the number of blocks contained within it are known, the agent may be enhanced with a number of simple heuristics. These heuristics assist the agent when interpreting data contained in short-term memory and then enhancing the map contained in long-term memory.

The deduction of the walls may be assisted by a number of simple rules. If one piece of wall is found, then the entire wall can be deduced. If a wall is found then we can establish the position of the wall running parallel to it.

When a block is discovered at location x, we can deduce that the walls can be no further than 5 squares in any direction, thus the 11x11 grid can be reduced in size. This heuristic has been named 'Smart Wall Deduction' (SWD) by the authors. Further analysis has resulted in the enhancement of SWD not only to use blocks but assume that a wall is never more than 5 squares from any explored square. The modified heuristic has been named Even-Smarter Wall Deduction (ESWD).

Once all 6 blocks have been found, any remaining memory locations marked as 'Something' must hold walls, and vice-versa once the entire wall has been discovered any remaining 'Something's must be blocks. This has been named the '6 block heuristic'.

4.2.4 The Genetic Algorithm

The genetic algorithm is used within the agent to evolve command sequences that may be carried out by the agent. Each chromosome consists of a list of commands in the form:

MMLMMMRM....

The commands are referred to as command sequences, and are interpreted thus:

M - Move forward 1 square

L - Rotate left

R - Rotate right

The length of the chromosomes was altered during the experiments carried out. Initially the chromosome length was set to 80, this being the maximum number of commands that may be carried before the agent runs out of energy.

Table 1: Chromosome Initialisation

Previous Genes	Possible values for current gene
L L	M
R R	M
L	L or M
R	R or M

Table 2: Initial fitness function rewards

Criterion	Reward
A block has just been pushed	3
A previously unknown square explored	2
A block has just been pushed into a wall	7

The GA is initialised with semi-random strings of genes. The authors identified a number of patterns that may occur within the chromosome that would result in the agent wasting energy (e.g. by rotating around in a circle). A simple initialisation scheme has been set up that restricts the choice of gene based on the previous genes (see Table 1). This scheme ensures that the initial population is free from wasteful patterns. Note that no repair occurs after mutation or crossover.

The recombination operator used is standard two-point crossover based on two parents creating one child. The mutation operator selects an individual with probability 0.1, a gene within that individual is then selected for mutation with the probability 0.02. The mutation consists of altering the value of the selected gene to M, L or R randomly.

A steady-state population of 500 is maintained. Selection and replacement of individuals will be facilitated by using a tournament selection operator. A tournament size of 7 was found to give reasonable results.

The fitness function evaluates the chromosome by simulating the execution of the command sequence using a copy of the map contained within long-term memory. The fitness function evaluates each command and rewards it based in the probable position of the agent after the command has been executed criterion as shown in Table 2.

After completing the route the final score (blocks against a wall + blocks in corners) is added to the fitness weighted by a factor of 100. Because the Tartarus problem has to be completed within a finite number of moves, the fitness function only examines those commands that could be executed given the remaining energy level.

5 Experiments

5.1 Experimental setup

Because of the deterministic nature of the GA used within the agent and the wide variety of starting configurations that exist for the Tartarus problem each experiment was carried out 100 times using randomly generated environments.

The software was initially implemented using ANSI standard C++, running on Redhat Linux. To allow for greater flexibility the software was subsequently re-written in Java. Later versions of the software were implemented across a 128 CPU parallel processing network.

5.2 The Initial Version

The initial version used a population size of 100 individuals, a mutation rate of 0.10 and a crossover rate of 0.10. Initially the GA was allowed to run until 1000 tournaments had been completed. Unless it is mentioned, it can be assumed that these basic parameters were used. The initial version incorporated no heuristics, and evaluated as many commands as the current energy level would allow. The average score achieved over 100 boards was 4.38. The distribution of scores was varied, one board scoring 8, four scoring 7 and the remaining 95% achieved scores of 6 or less.

Analysis of boards where the agent achieved a low score showed that a frequent problem was the agent pushes a block while unknown to the agent there's another block or a wall behind this block. In this case, the agent knows there's something behind this block, but it does not know whether this is a block or a piece of wall. Noting this in the long-term memory map would be useful, because the agent would be less likely to try and push this block. In the fitness evaluation (see Section 4.2.4), no points are gained for trying to push a block while knowing this is not possible. In order to be able to note down such information in long-term memory, the data type 'Something' (see Section 4.2.2) was added, allowing the average score over 100 boards to rise to 6.09.

With the addition of the initial SWD heuristic (as described in section 4.2.3) the average score was further increased to 6.21.

It was felt that the GA was running for too brief a period, and because there is no time constraint on the Tartarus problem, the authors allowed the GA to run

Table 3: Average scores over 100 boards using advanced edge detection, the six block heuristic and forcing a restart after hitting a known wall

ESWD	6-block	Restart After Wall	score
0	0	1	7.52
0	0	0	7.39
0	1	1	7.60
0	1	0	7.32
1	0	1	7.41
1	0	0	7.44
1	1	1	7.50
1	1	0	7.40

for 10,000 tournaments. To avoid premature convergence the population size was increased to 500. This modification caused the system to slow down, but the average score increased to 7.95. In the case of two boards the system managed to solve the Tartarus problem completely by achieving the maximum score possible (10).

5.3 Advanced Heuristics

Further analysis showed that the GA sometimes produced a command sequence that forced the agent to move forward into a wall. In our implementation, driving the agent into a wall halts execution of the command sequence and starts a new GA to evolve a new sequence. It was decided that although this move might appear to be illogical, the restarts might be unnecessary. The remainder of the command sequence may contain commands to solve the problem, and although energy might be wasted walking into a wall, a high overall score might be achieved. The effect of switching forcing restarts is shown in Table 3 (third column).

The '6 Block' heuristic and the ESWD heuristics (see section 4.2.3) have been implemented and the results obtained through their use can be seen in Table 3.

Reference to Table 3 allows us to draw the following conclusions, the best score was achieved using the 6-block heuristic, with the use of ESWD and allowing the GA to restart after the agent hits a wall.

The final scores achieved by the GA with the addition of the heuristics can be seen in Figure 2. The 'bump' at score 4 is accounted for by those instances where the GA has pushed 4 blocks together by accident in the beginning of the run. The largest distribution is at score 8, with a bell-like curve around it.

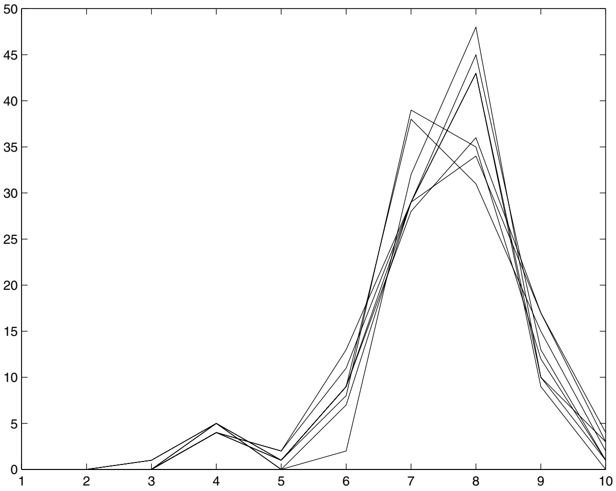


Figure 2: Score distributions for Table 3

5.4 Combining the GA with brute-force

When there is only a small amount of energy left, it is quicker for the system to perform an exhaustive search using every possible command sequence, rather than running the GA again. When the number of amount of remaining energy drops to below a given threshold, the system employees exhaustive search to finish the problem.

In Section 5.5 the exact number of legal strings is calculated for each length. If the number of strings examined by the GA (= #tournaments + population size) is more than the total number of legal strings, exhaustive search will take place.

The GA has always been allowed, so far to produce command strings that if fully executed would use up all the agents' remaining energy. It was felt that some improvement might be forthcoming if the GA was only allowed to produce small strings. This will not only concentrates the evolution into a smaller search space, but also reduces that amount of energy lost.

Table 4 shows the results when examining chromosome lengths between 7 and 20. The GA in figure 12 is also using the brute force method for calculating the final strings.

By only looking ahead a small number of moves (about 12) the scores rise up to 8.77. The reason for this improvement may be attributed to the fact that the GA almost never executes the last moves in the command sequence, while they do count in the fitness calulation.

Whilst starting to solve the problem, new information concerning the landscape will be frequently be found,

Table 4: Results for reducing the number of moves for the GA to look ahead. In column three, the average number of times the GA is run per board is shown. The average number of evaluations per board is the number of strings considered per board (= #runs * (populationsize + #tournaments)). The average number of actions per board is the number of actions (M,L,R) considered by the agent (=#evals * chromosomelength).

Len of chromo	Avg. score	#runs of GA	evals/ board	actions/ board
7	8.42	20.60	30900	216300
8	8.69	19.14	28710	229680
9	8.67	18.48	27720	249480
10	8.66	18.41	27615	276150
11	8.63	18.04	27016	297176
12	8.77	17.33	25995	320040
13	8.67	17.32	25980	337740
14	8.73	16.64	24960	359100
15	8.60	17.23	25845	387675
20	8.67	16.63	24945	498900

after only a few commands have been executed. It is wasteful and even misleading to include the later steps in the fitness function.

There should be an optimum number of moves to look ahead when evolving a command sequence. Too few moves will prevent the GA evolving a meaningful sequence, but too many moves are misleading.

The execution time of a board is typically between 3 and 5 minutes. Note that our system was not optimized for speed, that it was written in Java and ran on a fairly slow processor (Pentium 200 MHz).

5.5 Method for Calculating the Exact Number of Allowed Strings for a Given Length

There is a large number of inefficient command sequences, such as an LR sequence where the R reverses the effect of the L without any side-effect. All strings with LR, RL, LLL or RR in it (RR is equivalent to LL, thus redundant) are therefore not considered when doing an exhaustive search.

The number of 'legal' strings can be calculated as follows. After an M, what can follow is M, LLM, LM or RM. The rewrite rules are given in Figure 3.

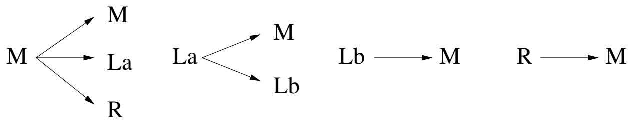


Figure 3: Legal strings

$La(x)$, $Lb(x)$, $R(x)$ and $M(x)$, i.e. the number of Las, Lbs, Rs and Ms at level x in the tree are calculated as follows:

$$\begin{aligned} La(x) &= M(x-1) \\ Lb(x) &= M(x-1) \\ R(x) &= M(x-1) \\ M(x) &= M(x-1) + La(x-1) + \\ &\quad + Lb(x-1) + R(x-1) \end{aligned}$$

with $M(0) = 1; La(0) = Lb(0) = R(0) = 0$. Level $x = 0$ is artificial, but with this initial setting all legal strings of length 1 and higher are correct.

5.6 A comparison to a non-evolutionary heuristic

Given the success of the exhaustive search in enhancing the GA, a full comparison of solving the Tartarus problem by replacing the GA with exhaustive search has been carried out. All the heuristics used to produce the data shown in Table 4 are still in use. The only difference is that instead of using an GA to evolve the command sequence using mutation and crossover, every possible command sequence generated using the rules in Section 5.5 is evaluated and the best taken as the command sequence. The maximum score presented in Table 5 (8.81) is slightly greater than that presented in Table 4 (8.77). An exhaustive search will usually always outperform an Genetic Algorithm, given the non-deterministic nature of the GA. Note though that for shorter lengths, the GA outperforms the exhaustive search, which is most likely due to the greater number of restarts of the GA. What is significant is the number of evaluations required per board, the exhaustive search evaluates 70% more command sequences for an overall gain of 0.5%. A comparison of the exhaustive search (look ahead length 14) and the GA (look ahaead length 12) may be seen in Figure 4. The exhaustive search method is especially good at scoring the maximum 10 points, while the GA score distribution peaks between 8 and 9. This would suggest that the exhaustive search is better at finding solutions to complete the problem that the GA, due to the exhaustive search always finding the optimal partial solution for the current board state. The exhaustive search heuristic performs best with a look

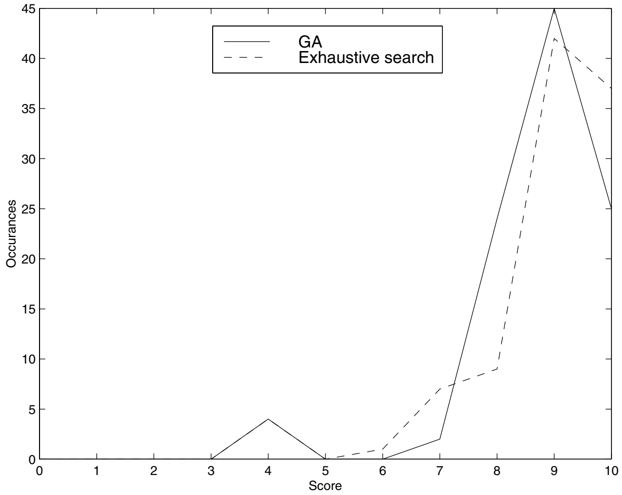


Figure 4: Score distributions for the GA with chromosome length 12 and the exhaustive heuristic with chromosome length 14

ahead of 14. This may be partly due to the fact that we have 80 energy points. If we assume that the algorithm produces stings of length l and restarts n times. The best performance will be recieved in situations where $n * l$ is equal to the energy level (ie all the moves in the final string can be executed). If we examine the nl relationship below we can deduce that a length of 14 with 6 restarts allows 10 out of 14 moves in the final string to be evaluated. Looking forward to the results in Table 5, we can see that indeed $l = 15$ performs worse than both $l = 16$ and $l = 14$.

```
6*12 = 72  7*12 = 84
6*13 = 78  7*13 = 91
5*14 = 70  6*14 = 84
5*15 = 75  6*15 = 90
4*16 = 64  5*16 = 80
```

Further research is needed to determine the exact relationship between chromosome length and the final result. A major problem is the unpredictablity of the number of runs of the GA. The number of runs is determined by the nature of the landscape that the agent is operating in.

5.7 Upscaling properties

In this section we will investigate how well our approach scales up to larger boards with more blocks. Following [Teller, 1994] we will use the following formulas for the number of pieces and the initial amount of energy:

Table 5: Results for reducing the number of moves for the exhaustive heuristic to look ahead. The number of valid command sequences is calculated as in Section 5.5. The last two columns are similar to those in Table 4.

Len	Avg. score	# runs	#valid com seq	evals/ board	actions/ board
1	0.84	80	3	240	240
2	0.96	40	6	240	480
3	3.44	27	13	352	1056
4	6.98	20	28	560	2240
5	7.39	16	60	960	4800
6	7.01	14	129	1806	10836
7	8.19	12	277	3324	23268
8	8.64	10	595	5950	47600
9	8.40	9	1278	11502	103518
10	8.65	8	2745	21960	219600
11	8.79	8	5896	47168	518848
12	8.81	7	12664	88684	1064208
13	8.76	7	27201	190407	2475291
14	8.91	6	58425	350550	4907700
15	8.57	6	125491	752946	11294190
16	8.78	5	269542	1437710	23003360

$$
\begin{aligned}
Pieces &= 1/3 * (N-2)^2 \\
Energy &= 2(N^2 + 2N - 3) - 10
\end{aligned}
$$

N is the width (and height) of the board. The -10 in the latter formula is somewhat artificial, but for reasons of comparability we will use it.

The results with chromosome length 12 are given in Table 6. Clearly the scores do not scale up terribly well. The reason for this is the (very) limited amount of initial energy, which makes initial exploration infeasible.

If we allow an initial energy of N^3, as argued in [Balakrishnan and Honavar, 1996], and make two more modifications, results are much better (see Table 7). Note that with larger boards, initial situations may occur that are partly unsolvable, e.g.

```
XX
X X
 XX
```

The modifications are:

- Make explorePoints a decreasing function of time.

Table 6: Results with chromosome length 12 for larger boards

N	Pieces	Energy	Max score	Average score
6	6	80	10	8.77
7	9	110	13	10.96
8	12	144	16	13.01
9	17	182	21	15.78
10	22	224	26	17.82

Table 7: Results with chromosome length 12 for larger boards with energy=N^3 and square penalty

N	Pieces	Energy	Max. score	Score	Energy used
6	6	216	10	9.23	113.38
7	9	343	13	12.17	146.20
8	12	512	16	14.95	206.52
9	17	729	21	19.55	290.75
10	22	1000	26	23.06	419.26

After some tuning we used the following formula:

$$ep = 2 + 10 \cdot e^{-4 * \frac{initialEnergy - energy}{initialEnergy}}$$

- Introduce a penalty for pushing a block into a known four block square. We used a very strong one: fitness = 0 if this happens.

6 Conclusions and future research

The authors have presented a novel approach to the Tartarus Problem. We have achieved the highest score in literature for the Tartarus Problem. An average score of 8.91 has been achieved by the exhaustive search heuristic with the fitness function introduced in this work.

The use of GA combined with the long-term memory gave an average result of 4.5, equivalent to that achieved using parse trees[Teller, 1994] and neural networks[Balakrishnan and Honavar, 1996]. The addition of heuristics to assist with the building of the long-term memory map such as smart wall deduction and the 6-block heuristic improved results. The most significant improvement, scoring 8.77, was achieved by the reduction in the length of the command sequence (chromosome).

Given the relative inefficiency of the exhaustive search, the hybrid GA approach developed by the authors would appear to be the most effective solution to the Tartarus problem yet published.

When allowed more initial energy, the agent scores close to optimal on all boards, even of larger sizes.

The basic agent developed here is now competent at solving the Tartarus problem. Future research may look at the possibilities of carrying out more complex tasks in similar environments. Although the fitness function and some of the heuristics used are specific to this problem, it remains to be seen whether the approach taken can be reapplied elsewhere.

Acknowledgements

We would like to thank Ernesto Costa and Jason Maassen for their helpful contributions to our work. This work was based on an problem set at the CoIL Summer School 2000. We would like to thank Adrian Trenaman for setting the problem.

References

[Ashlock and Freeman, 2000] Dan Ashlock and Jennifer Freeman. A pure finite state baseline for tartarus. In *CEC 2000*, volume 2, pages 1223–1230, 2000.

[Ashlock and Joenks, 1998] Dan Ashlock and Mark Joenks. ISAc lists, A different representation for program induction. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 3–10, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Ashlock and McRoberts, 1997] Dan Ashlock and McRoberts. A gp-automata reprise of astro teller's bulldozer experiment. Technical Report AM97-17, ISU Mathematics, 1997.

[Balakrishnan and Honavar, 1996] Karthik Balakrishnan and Vasant Honavar. On sensor evolution in robotics. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 455–460, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Teller, 1994] Astro Teller. The evolution of mental models. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 9, pages 199–219. MIT Press, 1994.

Reducing the Sampling Variance when Searching for Robust Solutions

Jürgen Branke
Institute AIFB, University of Karlsruhe
76128 Karlsruhe, Germany
Email: branke@aifb.uni-karlsruhe.de

Abstract

For real world problems it is often not sufficient to find solutions of high quality, but the solutions should also be robust. By robust it is meant that possible deviations from the solution should be tolerated, still yielding a good expected performance.
One way to reach this goal is to evaluate each individual several times under a number of different scenarios, taking the average performance as fitness. But although this method is effective, it requires significant computational power. In this paper, we continue some previous work aimed at minimizing the search effort while still providing the desired robustness. In particular, we examine the effectiveness of de-randomizing the sampling mechanism using variance reduction methods, and the question whether the same scenarios should be used for all individuals in the population or not. As will be shown, a significant performance gain can be obtained by taking these ideas into account, without any additional computational cost.

Keywords: evolutionary algorithm, robustness, sampling, noise

1 Introduction

In many real world situations adaptation to a changing environment is not possible, e.g. because the environment changes too quickly, because the environment cannot be monitored closely enough, or because the changes occur after the commitment to a particular solution has been made.

In such cases, i.e. when the solution can not be adapted, one might want to take precautions and search for a solution that performs well over all possible future scenarios that can be expected.

This property of being insensitive to slight changes in the environment or noise in the decision variables is also called *robustness*.

Some examples for applications include:

- In manufacturing, it is often not possible to produce a product exactly according to the design specifications, but instead, certain manufacturing tolerances have to be allowed. A product design should take this into account and yield good results despite of these tolerances.
- In scheduling problems, solutions are sought that allow for e.g. slight variations in processing times, time for machine breakdowns or the incorporation of an additional urgent job without requiring a total reordering of the production plan.
- For many control problems, the environment may change slowly, e.g. machines may slowly wear out or the composition/quality of the raw material may change slightly. Often it is not possible to constantly monitor and control the process, thus it is advantageous to implement solutions that yield good results over a certain range of environmental conditions.

Due to the uncertainty, the fitness function in those cases is stochastic. If $f(x, e)$ is the fitness function determining the quality of a solution x in an environment e, then in the example of manufacturing tolerances, prior to evaluation, the solution x (the product design) may be disturbed, i.e. $x \mapsto x + \delta$. Not the desired product x, but some variant $x + \delta$ is produced. Similarly, in the other two examples, the environment may be disturbed, i.e. $e \mapsto e + \delta$.

Table 1: Terminology of fitness values used

	Name	Meaning
$f(x)$	raw fitness	Function that can be calculated deterministically
$f_{eff}(x)$	effective fitness	Actual optimization criterion, implicitly defined by f and the probability distribution of disturbances. Assumed to be unavailable to the optimization algorithm
$f_{est}(x)$	estimated fitness	Estimated effective fitness used by the EA. Estimation is usually done on the basis of disturbed evaluations $f(x+\delta)$

In this paper, we will restrict our attention to the case of a deterministic environment and a disturbed solution, i.e. $f = f_e(x+\delta)$ (just like in the case of manufacturing tolerances). However, all presented results can be readily transferred to the case of a disturbed environment as well.

We define a robust solution as the solution that maximizes the expected performance. This expected performance will also be called *effective* fitness $f_{eff}(x)$ and, given a probability density function $p(\delta)$ of disturbances δ, could in principal be calculated as

$$f_{eff}(x) \quad = \quad E(f(x+\delta)) = \int_{-\infty}^{\infty} p(\delta) \cdot f(x+\delta)\, d\delta$$

Of course, for problems of relevant complexity, this calculation will not be possible because the function f is not known in closed form but perhaps only given in the form of a simulation model. Consequently, $f_{eff}(x)$ has to be estimated from single evaluations of $f(x+\delta)$.

To estimate the expected average quality of an individual, one straightforward possibility is to probe the fitness landscape at several random points in the neighborhood and calculate the mean. This average, denoted f_{est} as opposed to the exact effective fitness f_{eff} may then be used by the EA as that individual's fitness. The effect is a smoothing of the fitness landscape, with sharp peaks being flattened and smooth hills being almost non-affected.

A major drawback of this method is the increased computation time: since usually evaluation is the time determining factor, evaluating several points for one individual is very expensive. Theoretically (cf. [TG97, Bra98]), because the EA frequently re-samples promising areas of the search space, it may be sufficient to use single disturbed samples for evaluation. However, using n samples for a single evaluation reduces the standard deviation of an individual's fitness distribution by a factor of $\sqrt{n}$, facilitating EA convergence. In this paper, we test the effectiveness of de-randomizing the sampling by using variance reduction methods. Furthermore, we test whether the scenarios should be the same for all individuals in the population, or whether they should be drawn independently.

The paper is structured as follows: In the next section, we give a comprehensive survey of earlier work in the area. Then, in Section 3, the different variance reduction methods are presented. Section 4 reports on the empirical results. The paper concludes with a summary in Section 5.

2 Related Work

The idea to consider the sensitivity of a solution besides its quality, and to specifically search for robust solutions, has appealed to a number of authors, and several different approaches can be found in the literature.

The method suggested by Parmee et al. [Par96a, Par96b, PJB94] involves searching for several high performance regions in the fitness landscape, and then examining each of these regions more closely e.g. in terms of their sensitivity to parameter changes. After that, further search can be restricted to those regions that are most promising in terms of robustness. Since only some small regions are assessed extensively, and then the search is restricted to areas that promise sufficiently robust solutions, the computational overhead is comparatively small. However, the approach does not optimize the desired criteria directly and may thus not yield optimal results. Furthermore, as Wiesmann et al. show in [WHB98], the optimal solution in terms of robust quality does not necessarily lie in a high performance region.

Leon et al. [LWS94] developed a problem-specific, explicit measure of robustness for job shop scheduling with machine breakdowns, which is then used to calculate the fitness values. However, since the estimate is very problem specific, this idea can not be readily transferred to other applications.

Hart et al. [HRN98] propose to evolve artificial immune systems, basically building blocks of solutions which, if combined appropriately, should be able to cover a wide range of disturbances. To achieve this goal, the artificial immune systems are tested against a random sample of possible disturbances. Of course, the basic idea here is to be able to quickly generate a new schedule to a new scenario, given the building blocks in the immune system. But apart from that, the authors show in [HRN98] that the solutions generated in that way are generally more similar to the original solution than if each solution were optimized from scratch, thus providing some robustness in the sense that the necessary changes are minimized. The authors speculate that it might also be possible to evolve single schedules covering many or all possible disturbances, but this idea has been left to future work. A similar approach has been suggested in [HR99b, HR99a].

The most general approach, which is also the approach discussed in more detail in this paper, is to evaluate an individual under several scenarios (i.e. to deliberately disturb evaluations) and to use the average performance as fitness measure. The resulting fitness function is stochastic, but due to their use of a search population and the repeated re-sampling of promising areas of the search space, evolutionary algorithms (EAs) are generally perceived to be well suited for optimization in noisy environments. This approach has already proven successful in a number of different applications like producing fault tolerant neural networks [SF92], evolving robot control systems [Jak97, NB97], the search for electronic circuits insensitive to temperature changes [Tho96, Tho98], flight control under changing conditions [Bly98], multilayer optical coatings insensitive to manufacturing tolerances [Gre94, Gre96, WHB98], wing-box optimization taking into account manufacturing tolerances [MHI96], or job shop scheduling [Ree92, TJH99, Ven98].

But although this method seems to be quite effective, it requires substantial computational power, since every individual has to be evaluated several times.

Tsutsui and Gosh [TGF96, TG97] were the first to examine the idea of disturbed evaluations more closely and showed, using the schema theorem, that given an infinitely large population size, an EA with single disturbed evaluations is actually performing as if it would work on the effective fitness function.

If the input to the evaluation function (i.e. the individual) is disturbed randomly for every evaluation, the *expected* returned value at every point in the search space is just equivalent to the effective fitness. Since in EAs, promising areas are probed quite often, this seems to be sufficient. Additionally, evaluation results in flat areas will be more consistent over time thus it could be easier for the EA to focus on these areas.

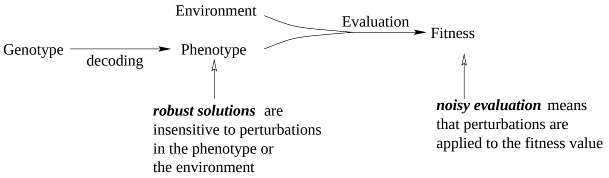

Environment
Evaluation
Fitness
Genotype
decoding
Phenotype
robust solutions are insensitive to perturbations in the phenotype or the environment
noisy evaluation means that perturbations are applied to the fitness value

Figure 1: Robust solutions vs. noisy fitness evaluation

Branke [Bra98, Bra00] has examined a number of ways to improve the EA performance without increasing the number of samples per individual, e.g. by taking the fitness of neighboring individuals into account, by increasing the population size, or by varying the number of samples throughout the run.

Note that the problem of creating robust solutions as defined here has some similarities to optimizing noisy functions, which have also been examined in combination with EAs. Papers in that area include for example [AW94, Arn00, Ang95, FG88, HB94, Mil97, Sta98]. However there are two main differences:

- with noisy functions, some noise is usually added to the output (quality) of the solution. In the settings regarded here, noise is added to the decision variables (or phenotype) of the solution. I.e. if $f(x)$ is the fitness function and δ is some (e.g. normally distributed) noise, then a noisy fitness function would mean $f'(x) = f(x)+\delta$, while in our case $f'(x) = f(x + \delta)$ (cf. Fig. 1 for illustration). From a practical point of view, the noise for noisy functions is usually considered to be symmetric and independent of the individual's location in the search space. On the other hand, if the input to the evaluation function is disturbed, naturally the observed distribution of fitness values for an individual is skewed, has non-zero mean and strongly depends on the location of the individual in the search space.
- noisy functions can not be evaluated without noise, the challenge for the EA is to find good solutions despite the noise. In the settings considered in this paper, it is assumed that only the decision variables of the final solution are necessarily disturbed, usual function evaluations during the EA run could also be performed without disturbance. This is justified because within an EA, evaluation is usually done on the basis of a theoretical, computerized model, while the final solution is then actually implemented and has to face all the uncertainties present in reality. Note that the ideas presented in this paper would not be practicable without this property.

These two aspects, optimization under noisy environments and the search for robust solutions, may be unified by making the distribution of the noise also dependent on x, i.e. $f'(x) = f(x) + \delta(x)$. Both former aspects are mathematically just special cases of the latter, however for the purpose of optimization, the knowledge whether the problem is of one type or the other may be exploited (like e.g. in this paper).

3 Variance Reduction Methods

As has been noted in Section 2, for the problem of searching for robust solutions that is regarded here, it is possible to accurately determine $f(x)$ during the EA run, which in turn allows to explicitly choose the disturbances δ used to evaluate the individuals. Generally, these disturbances have been drawn randomly according to the probability distribution that is expected in reality.

However, by using more advanced sampling methods, the variance of the distribution of fitness values for a single individual may be reduced, and thus the EA's search capabilities can be enhanced.

The first issue that will be addressed in this section is whether the disturbances used for evaluation should be the same for all individuals in a population (denoted as *population*), or independent from individual to individual (denoted as *single*). Using the same disturbances throughout the run would not make sense, since then the EA could concentrate on exploiting the structure of the fixed set of disturbances.

Using different disturbances for different individuals of the same generation might be considered unfair, because a good individual evaluated under unlucky circumstances may be ranked worse than a bad individual with a lucky evaluation. However, over the long run, this should cancel out, as high performance regions are sampled many times (as has already been argued in [NB97]). Also, the alternative, namely using the same disturbances for all individuals of one generation, may be just as unfair, because disturbances that lead to above-average evaluations for one individual may lead to below-average evaluations for another individual. An example is given in Figure 2: With individuals sitting on both peaks, a shift to the right might push all individuals on the left peak over the cliff, while the individuals on the right peak are only slightly affected.

The second question addressed in this section is how to select the different disturbances for multiple evaluations of a single individual such that the variance is minimized. This is very similar to the problem of Monte Carlo Integration (cf. e.g.[Gen98]) and also related to experimental design [Hic93] and simulation [Rip87]. However, compared to the just mentioned areas, the number of samples that can be taken in our case is very small, usually smaller than the number of dimensions, because thousands of individuals have to be evaluated during the course of the EA.

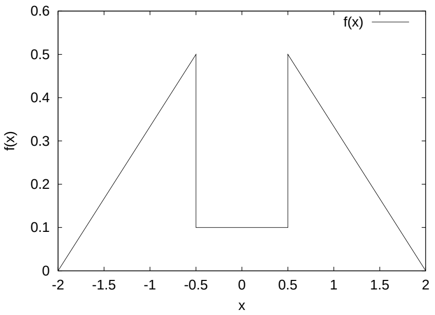


Figure 2: If there are individuals on both peaks, using the same disturbance for all individuals will favor one peak over the other.

In the following experiments, four sampling methods are compared:

1. *Random:* Each disturbance is chosen independently at random (this is the default used for most experiments).
2. *Antithetic:* Always produces pairs of disturbances which lead to negatively correlated estimations. For uniformly distributed disturbances, the first disturbance vector $\vec{\delta}$ is chosen at random, the second is then chosen as $-\vec{\delta}$. For more details see also [Gen98, Rip87].
3. *Stratified:* Stratified sampling divides the space of possible disturbances into regions of equal probability and draws one disturbance from every region. In the implementation used here, some dimensions are divided into positive and negative values. If, for example, 4 samples are needed, then the first two dimensions are divided, and the samples are drawn with the signs $(+/+)$, $(+,-)$,$(-,+)$, and $(-,-)$ for the first two dimensions respectively. For more details on stratified sampling see e.g. [Gen98, Rip87].
4. *Latin Hypercube:* here, in order to draw k samples, the range of disturbances in each dimension is divided into k equal parts, and a random sample is chosen from each part, resulting in k values for each dimension. These values from the different dimensions are then combined randomly to form the k disturbances. As

an example, consider the generation of 4 disturbances with 3 dimensions. For the uniformly distributed disturbance in the interval $[-0.2, 0.2]$ used throughout this paper, for each dimension i, four values $d_i^1 \in [-0.2, -0.1], d_i^2 \in [-0.1, 0.0], d_i^3 \in [0.0, 0.1], d_i^4 \in [0.1, 0.2]$ are randomly chosen. Then, these are combined to form the samples $(d_1^{\pi_1(j)}, d_2^{\pi_2(j)}, d_3^{\pi_3(j)})$ with $j = 1 \dots k$ and π_i being a random permutation of the values for dimension i. The advantage of latin hypercube sampling is that in each of the k subranges in each of the n dimensions, there is exactly one sample. Note that this method has to be slightly modified if the disturbance is not uniformly distributed. Then, the intervals for each dimension should be adapted such that the probability of a random disturbance δ to fall into interval i is equal for all i. Also, the random sample taken from that interval should be taken according to the probability distribution. See also [MCB79, Gen98].

The four sampling methods described above are illustrated by an example in Figure 3.

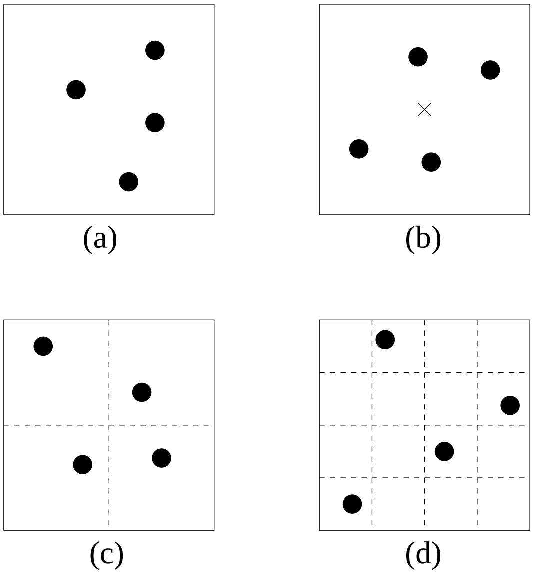


Figure 3: Examples for drawing 4 samples in a 2-dimensional rectangular area using random sampling (a), antithetic sampling (b), stratified sampling (c), and latin hypercube sampling (d).

4 Empirical Evaluation

4.1 Test Problems

Due to computational efficiency and for the sake of easy analysis, simple mathematical functions are used for testing the effectiveness of the different approaches.

Also for reasons of computational efficiency, in the experiments reported below, a uniformly distributed disturbance δ is used. Note, however, that all suggested approaches could be readily applied to any kind of disturbance.

For every test function f_i, we give the exact definition for a single dimension, denoted by $\hat{f}_i$. These are scaled to n dimensions by simple summation over all dimensions, i.e.

$$f_i(x) = \sum_{j=1}^{n} \hat{f}_i(x_j)$$

In the experiments reported below, 20 dimensions were used.

The probability density function of the disturbance δ is assumed to be uniform over the interval $[-0.2; 0.2]$ in each dimension, and independent of the test function.

4.1.1 Test Function f_1

Function f_1 basically offers two main alternatives in each dimension: a high sharp peak at $x = 1.0$ and a smaller not so sharp peak at $x = -1.0$. The sharp peak would be preferable in a deterministic setting, but the other peak actually has a higher effective fitness. In order to avoid an initialization bias, both hills' basins of attraction are of the same size and have the same average quality (i.e. the x-intervals and the areas below the curve are the same on both sides). This basic function is overlayed with a strong low frequency oscillating function to make optimization more challenging.

The mathematical formulation of this function may be given as

$$\hat{f}_1(x_j) = \begin{cases} -(x_j+1)^2 + 1.4 - 0.8|\sin(6.283 \cdot x_j)| & : -2 \leq x_j < 0 \\ 0.6 \cdot 2^{-8|x_j - 1|} + 0.958887 \\ \quad -0.8|\sin(6.283 \cdot x_j)| & : 0 \leq x_j < 2 \\ 0 & : \text{otherwise} \end{cases}$$

$$x_j \in [-2; 2]$$

Figure 4 provides visualizations of f_1 and $f_{1,eff}$ in one dimension. For the 20 dimensions used, the maximal effective fitness is $f^*_{1,eff}(x) \approx 18.93$ at $x_i = -1$.

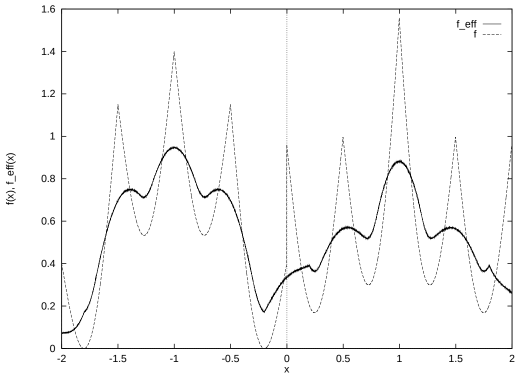


Figure 4: Test function f_1 and effective fitness $f_{1,eff}$ when disturbed.

4.1.2 Test Function f_2

Function f_2 is designed to show the influence of asymmetry on the optimization behavior. Again, the function has only a single peak, but this time it drops sharply on one side (cf. Figure 5). The maximum for $f_{2,eff}(x)$ is at $x = 0.0$, for the 20 dimensions used, the maximal effective fitness is $f^*_{2,eff}(x) = 16.0$. This function allows to rate the approaches according to their risk awareness: the further they approach the peak, the greater their acceptance of the "risk" to drop down the edge in case of disturbances.

$$\hat{f}_2(x_j) = \begin{cases} x_j + 0.8 & : \quad -0.8 \leq x_j < 0.2 \\ 0 & : \quad \text{otherwise} \end{cases}$$
$$x_j \in [-1;1]$$

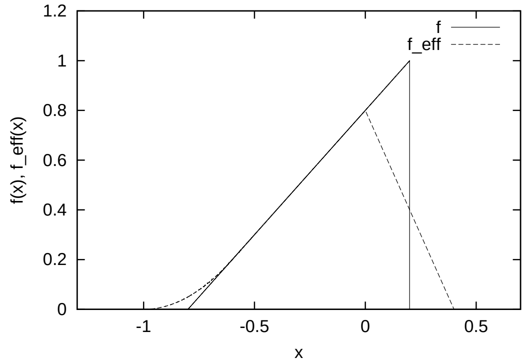


Figure 5: Test function f_2 and effective fitness $f_{2,eff}$ when disturbed.

4.2 Experimental Setup and Default Parameters

Since the differences in the algorithms compared are relatively small, the parameter settings are not expected to have too much influence on the relative performances. Therefore, it has not been tried to tune the parameters for every algorithmic variant. Instead, for all algorithms compared in one set of experiments, always the same parameter settings are used.

Unless stated otherwise, the default settings were a population size of 50, generational reproduction without elitism, and two-point-crossover with probability of 0.6. For mutation, with a probability of $\frac{1}{n}$ (n: chromosome length), an allele a_i is altered by adding a Gaussian random variable, i.e. $a_i \mapsto a_i + \delta$ with $\delta \in N(0, \frac{2}{15})$. The selection bias b for ranking selection was set to 2.0.

The number of evaluations or samples used to calculate the average fitness of a single individual is denoted *sample size* . In the experiments reported below, the sample size has been set to 4.

As performance measure, the true effective fitness of the single final solution as returned by the optimization algorithm is considered. Since the effective fitness is assumed to be unknown to the algorithm, this final solution is chosen as the best individual from the final population with respect to the average of 100 additional disturbed evaluations for each individual in the final population. The actual effective fitness for that individual is then determined externally and used as performance measure.

To reduce the influence of randomness, all reported results are averaged over 50 runs with different random seeds. To determine whether the performance of two algorithms differs significantly, a two-sided t-test for unknown variances (Fisher-Benders Test) and error probability of 0.05 has been used.

4.3 Results

In order to assess the quality of the above methods, each sampling method has been implemented twice, once drawing a different disturbance for every individual, and once using the same disturbances for all individuals in the population.

In all cases, the EA was run for 200,000 evaluations. The results are depicted in Table 2 for test function f_1 and Table 3 for test function f_2.

Obviously, for the settings tested, amongst the different sampling methods, only the latin hypercube sampling yields significant improvements. Comparing the use of different disturbances for each individual or the same disturbance for all individuals of one population, the latter seems to work better. Overall, by using latin hypercube sampling and the same disturbances for all

Table 2: Comparison of different sampling strategies for sample size 4 and test function f_1. Results marked with † differ significantly from the random/single strategy as determined by a two-sided t-test and error probability of 0.05.

	single	population	avg
random	17.94	18.05†	18.00
antithetic	17.95	18.12†	18.04
stratified	17.93	18.05†	17.99
latin hypercube	18.18†	18.30†	18.24
avg	18.00	18.13	18.07

Table 3: Comparison of different sampling strategies for sample size 4 and test function f_2. Results marked with † differ significantly from the random/single strategy as determined by a two-sided t-test and error probability of 0.05.

	single	population	avg
random	15.41	15.48†	15.45
antithetic	15.46	15.49†	15.48
stratified	15.41	15.46	15.45
latin hypercube	15.52†	15.53†	15.53
avg	15.45	15.49	15.47

individuals of one population, the performance could be improved from 17.94 to 18.30 respectively from 15.41 to 15.53 when compared to the standard method of using random independent disturbances for each individual. This improvement is particularly noteworthy because it may be achieved without any additional function evaluations. In fact, since fewer random numbers have to be generated, the computational cost is actually slightly smaller.

5 Summary

Often, in real world situations, the goal is to find *robust* solutions, i.e. solutions that perform well over a wide range of possible scenarios. Instead of a solution with optimal fitness, a solution with optimal *effective* fitness is sought, which has been defined as the average fitness over all possible scenarios.

An important application is the case of manufacturing tolerances. A product design should be robust in the sense that in spite of the tolerances, the quality of the manufactured items is reliably high.

One standard way to cope with the inherent uncertainties is to evaluate an individual by using the average of a number of samples. In this paper, it has been examined how the estimation variance can be reduced by de-randomizing the sampling. Also, the alternatives of using the same disturbances for all individuals in the population as opposed to independent disturbances for each individual have been tested.

As has been shown by empirical tests, the most successful approach was to use latin hypercube sampling instead of random sampling. Some additional benefit could be gained by using the same disturbances for all individuals of the population. These ideas combined significantly improved the performance while actually slightly reducing the computational load.

References

[Ang95] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence: A Dynamic System Perspective*, chapter 11, pages 152–163. 1995.

[Arn00] D. Arnold. Evolution strategies in noisy environments - a survey of existing work. In *Second EvoNet Summer School on Theoretical Aspects of Evolutionary Ccomputing*. Springer, 2000.

[AW94] A. N. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, pages 97–122, 1994.

[Bly98] Philip W. Blythe. Evolving robust strategies for autonomous flight : A challenge to optimal control theory. In Ian Parmee, editor, *Adaptive Computing in Design and Manufacture*, pages 269 – 283. Springer - Verlag London, 1998.

[Bra98] J. Branke. Creating robust solutions by means of an evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, number 1498 in LNCS. Springer, 1998.

[Bra00] J. Branke. Efficient evolutionary algorithms for searching robust solutions. In I.C. Parmee, editor, *Adaptive Computing in Design and Manufacture*, pages 275–286. Springer, 2000.

[FG88] J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.

[Gen98] J. E. Gentle. *Random Number Generation and Monte Carlo Methods.* Springer, 1998.

[Gre94] H. Greiner. Robust filter design by stochastic optimization. In F. Abeles, editor, *Optical Interference Coatings, Proc. SPIE*, pages 150–161, 1994.

[Gre96] H. Greiner. Robust optical coating design with evolutionary strategies. *Applied Optics*, 35(28):5477–5483, 1996.

[HB94] U. Hammel and T. Bäck. Evolution strategies on noisy functions, how to improve convergence properties. In Y. Davidor, H. P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature*, number 866 in LNCS. Springer, 1994.

[Hic93] C. R. Hicks. *Fundamental Concepts in the Design of Experiments.* Oxford University Press, 4 edition, 1993.

[HR99a] E. Hart and P. Ross. The evolution and analysis of a potential antibody library for use in jobshop scheduling. In D. Corne et al., editor, *New Ideas in Optimization*, chapter 12, pages 185–202. McGraw Hill, 1999.

[HR99b] Emma Hart and Peter Ross. An immune system approach to scheduling in changing environments. In *Genetic and Evolutionary Computation Conference*, pages 1559–1566. Morgan Kaufmann, 1999.

[HRN98] E. Hart, P. Ross, and J. Nelson. Producint robust schedules via an artificial immune system. In *International Conference on Evolutionary Computation*, pages 464–469. IEEE, 1998.

[Jak97] N. Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6(2):325–368, 1997.

[LWS94] V. J. Leon, S. D. Wu, and R. H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, September 1994.

[MCB79] M. D. McKay, W. J. Conover, and R. J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[MHI96] M. McIlhagga, P. Husbands, and R. Ives. A comparison of search techniques on a wing-box optimisation problem. In H.-M. Voigt, editor, *Parallel Problem Solving from Nature 4*, number 1141 in LNCS, pages 614–623. Springer, 1996.

[Mil97] Brad L. Miller. *Noise, Sampling, and Efficient Genetic Algorithms.* PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1997. available as TR 97001.

[NB97] P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107–140, 1997.

[Par96a] I. C. Parmee. Cluster-oriented genetic algorithms for the identification of high-performance regions of design spaces. In *Proceedings of EvCA96*, 1996.

[Par96b] I. C. Parmee. The maintenance of search diversity for effective design space decomposition using cluster-oriented genetic algorithms (cogas) and multi-agent strategies (gaant). In *Proceedings of ACEDC'96*, 1996.

[PJB94] I. C. Parmee, M. Johnson, and S. Burt. Techniques to aid global search in engineering design. In *Proceedings of International Conference on Industrial and Engineering Applications of AI and Expert Systems*, 1994.

[Ree92] C. R. Reeves. A genetic algorithm approach to stochastic flowshop sequencing. In *Proceedings of the IEE Colloquium on Genetic Algorithms for Control and Systems Engineering*, number 1992/106 in Digest, pages 13/1–13/4. IEE, London, 1992.

[Rip87] B. D. Ripley. *Stochastic Simulation.* John Wiley, 1987.

[SF92] A.V. Sebald and D.B. Fogel. Design of fault tolerant neural networks for pattern classification. In D.B. Fogel and W. Atmar, editors, *1st Annual Conference on Evolutionary Programming*, pages 90–99, San Diego, 1992. Evolutionary Programming Society.

[Sta98] P. Stagge. Averaging efficiently in the presence of noise. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature V*, volume 1498 of *LNCS*, pages 188–197. Springer, 1998.

[TG97] S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–208, 1997.

[TGF96] S. Tsutsui, A. Ghosh, and Y. Fujimoto. A robust solution searching scheme in genetic search. Number 1141 in LNCS, pages 543–552. Springer Verlag Berlin, 1996.

[Tho96] A. Thompson. Evolutionary techniques for fault tolerance. In *Proc. UKACC Intl. Conf. on Control*, pages 693–698. IEE Conference Publications, 1996.

[Tho98] A. Thompson. On the automatic design of robust elektronics through artificial evolution. In A. Peres-Urike M. Sipper, D. Mange, editor, *Proceedings of the 2nd International Conference on Evolvable Systems*, pages 13 – 24. Springer, 1998.

[TJH99] M. Tjornfelt-Jensen and T. K. Hansen. Robust solutions to job shop problems. In *Congress on Evolutionary Computation*, volume 2, pages 1138–1144. IEEE, 1999.

[Ven98] C. Ventouris. Gestaltung robuster Maschinenbelegungspläne unter Verwendung evolutionärer Algorithmen. Master's thesis, Institute AIFB, University of Karlsruhe, 1998.

[WHB98] D. Wiesmann, U. Hammel, and T. Bäck. Robust design of multilayer optical coatings by means of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2(4):162–167, 1998.

Efficient Fitness Estimation in Noisy Environments

Jürgen Branke, Christian Schmidt, Hartmut Schmeck
Institute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
Phone: ++49(721)608-6585
Email:{branke—schmeck}@aifb.uni-karlsruhe.de

Abstract

In noisy environments, an individual's fitness cannot be evaluated precisely, but its fitness has to be estimated. Evolutionary Algorithms are generally believed to be able to cope well with a stochastic fitness function because promising areas of the search space are sampled several times, thus an estimation error has limited effect. Nevertheless, methods to improve the estimation accuracy should help to speed up evolution. In this paper, we suggest to use local regression for estimation, taking the fitness of neighboring individuals into account. As a side-product of this approach, we also obtain information on the gradient of the fitness landscape, which may then additionally be used to perform local hill-climbing.

Keywords: evolutionary algorithm, noise, regression, hill-climber

1 Introduction

In some applications, an individual's fitness cannot be determined precisely, but is somehow disturbed. However, due to their use of a search population of individuals and the repeated re-sampling of promising areas of the search space, evolutionary algorithms (EAs) are generally perceived to be well suited for optimization in such environments.

There are basically two major areas of research that deal with such difficulties:

1. Optimization in noisy environments
2. Search for robust solutions.

In noisy environments, the fitness of the solution is disturbed, i.e. if $f(x)$ is the fitness function and δ is some (e.g. normally distributed) noise, then a noisy fitness function would mean $F(x) = f(x) + \delta$ and the maximum of $f(x)$ is sought despite the noise. Papers in that area include for example [1, 2, 3, 6, 7, 9, 14].

When searching for robust solutions, the actual solution, or the input to the fitness function is disturbed, i.e. $F(x) = f(x + \delta)$, and a solution with good expected performance given the distribution of δ is sought. Examples for that research can be found e.g. in [4, 10, 11, 15, 16], a brief survey can be found in [5].

A major difference between the two areas above may be that while for noisy environments, the fitness noise is usually assumed to be independent of the location of the individual, for the search of robust solutions, the distribution of fitness values at a specific location depends on the shape of the fitness function at that location and is therefore different in different areas of the search space. Furthermore, the distribution is typically skewed and has non-zero mean.

In principle, the two cases may be unified by making the distribution of noise also dependent on x, i.e. $F(x) = f(x) + \delta(x)$. Optimization under noisy environments and the search for robust solutions are mathematically just special cases with particular distributions of $\delta(x)$.

There are some additional differences of the two research areas, but nevertheless they are related.

In both cases, the influence of δ may be reduced by taking a number of samples for each individual and using the average as fitness for the individual. A sample size of n decreases the standard deviation of the distribution of fitness values by a factor of $1/\sqrt{n}$, but it increases the number of required fitness evaluations by a factor of n. Thus, this simple approach is not applicable when fitness evaluations are costly.

This paper focuses on the idea of reducing the noise in the fitness evaluation of an individual x^* by taking the fitness evaluations of other, previously evaluated individuals in the neighborhood into account. This should help to increase the estimation accuracy without requiring additional fitness evaluations.

This general idea has first been suggested and implemented in a rather straightforward manner in [4]. Sano and Kita [12, 13] examined the idea more closely and used more rigorous statistical models for estimation. In this paper, we will further elaborate on the idea, using local regression and model adaptation. Futhermore, since by using local regression we implicitly also obtain local gradient information on the search space, we suggest to use this information to integrate a local hillclimber.

For the remainder of this paper, we restrict our attention to the case of noisy function evaluation, although we also look at a function with different noise in different regions of the search space, thus we think that the results will probably apply to robust solution searching as well.

The paper is organized as follows. In the following section, we will explain and motivate the idea of using local regression. The approch will be compared to earlier related approaches in Section 3. The integrated hill-climber is then discussed in Section 4. An empirical comparison of the different methods can be found in Section 5. The paper concludes with summary and outlook in Section 6.

2 Statistical Model

Let $f(x)$ be the fitness function to be optimized, and $F(x) = f(x) + \delta(x)$ designate the actual samples.

We base our model on the following three assumptions:

1. $f(x)$ may be locally approximated by a low polynomial function (e.g. constant, linear, or quadratic)
2. The variance within a local neighborhood is constant
3. The noise is normally distributed, i.e. $\delta(x) \sim N(0, \sigma^2(x))$

These assumptions then allow to use local regression for estimation.

The idea of local regression is to consider only those data points for which Assumptions 1-3 are valid. For that purpose, a weighting function $w_h(d)$ is used to determine the influence of a data point y to the estimation at location x^*, with decreasing weight for increasing distance d between x^* and y. A typical weighting function is for example the tri-cube function [8]:

$$w_h(d) = \begin{cases} (1 - \frac{d}{h}^3)^3 & : \; d < h \\ 0 & : \; otherwise \end{cases}$$

with h being a parameter defining the size of the local neighborhood.

According to this model, evaluations of an individual y in the neighborhood of a particular individual x^* is approximately distributed as

$$F(y) \sim N(g(x^*), \sigma(x^*))$$

with g being a low polynomial function. The validity of this approximation shrinks with increasing distance d between x^* and y, which is reflected in the weighting function $w_h(d)$ used for regression.

The model is visualized in Fig. 1.

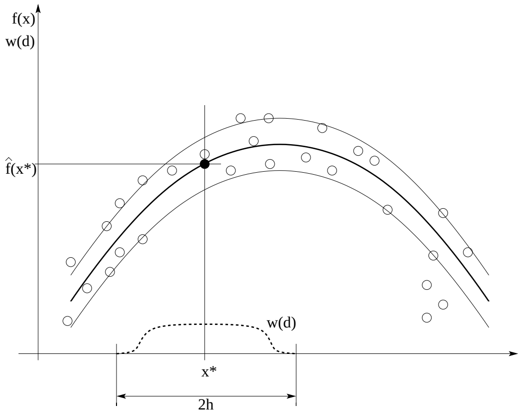


Figure 1: Model of the search space. Displayed is the estimated quadratic fitness function plus/minus constant variance. The validity of the regression shrinks with increasing distance from x^* as indicated by the weighting function $w_h(d)$.

An important issue of the model is the proper estimation of the neighborhood parameter h, reflecting the size of the neighborhood for which the above assumptions are valid.

In this paper, we are testing two approaches to setting the parameter h.

In the *relative neighborhood* model, h is chosen as the smallest distance such that at least 5% of all available samples are in the considered neighborhood. Since the EA will continue to generate new samples in high performance regions, h will become smaller over the course of the run, while the number of samples used increases, leading to increased accuracy.

As an alternative, in the *adaptive neighborhood* model, h is optimized for every fitness value to be estimated.

In [8], a number of methods are suggested to optimize h. Here, we decided to use the Cross-validation criterion, which is defined as:

$$CV(\hat{g}_{x^*,h}) = \frac{\sum_{i=1}^{n} w_h(d_i)(F(x_i) - \hat{g}^{-i}_{x^*,h}(x_i))^2}{\sum_{i=1}^{n} w_h(d_i)}$$

with d_i being the distance between individuals x^* and x_i, $\hat{g}^{-i}_{x^*,h}$ being the regression function around location x^* estimated using neighborhood-parameter h, not taking into account the sample x_i.

Basically, CV is the weighted average of errors that occur when sample x_i is estimated from all other samples, in turn for all samples i.

The neighborhood-parameter h is then chosen such that CV is minimized. For minimization, a simple numerical hill climber is used.

Note that in order to estimate a quadratic (linear, constant) fitness function in m dimensions, at least $1+m+m(m+1)/2$ (respectively $1+m, 1$) linearly independent samples at different locations are necessary. Thus h has to be at least large enough to ensure that the neighborhood encompasses the required number of samples.

Once a proper h has been determined, the fitness $f(x)$ can be estimated by using local regression:

$$\hat{f}(x^*) = \hat{g}_{x^*,h}(x^*)$$

Note that the computational complexity of the approach is substantial and strongly dependent on the number of dimensions of the problem. As has been illustrated above, for quadratic regression, $v = O(m^2)$ variables have to be estimated in m dimensions, and the regression then has complexity $O(nv^2)$, with n being the number of individuals in the neighborhood (bounded by the number of data points in the history).

This complexity could potentially be reduced either by

- using linear instead of quadratic regression
- introducing some a priori knowledge about dimension dependencies, effectively reducing the necessary number of variables that have to be estimated
- restricting the neighborhood size and thus the number of individuals that are considered for regression.

Nevertheless, in Section 5 we compare the approaches solely on the basis of calls to the evaluation function, assuming that the evaluation function is relatively complex, e.g. involving a finite element simulation.

3 Comparison with Earlier Approaches

As has been noted in the introduction, similar approaches have already been examined in [4, 12, 13]. We try to point out the differences using the terminology introduced above.

The differences between the three estimation methods are summarized in Table 1.

In [4], it was suggested to estimate the fitness as weighted average with a linearly decreasing weight function up to some fixed maximum distance d_{max}. This corresponds to estimating a constant by regression, taking into account only samples with distance $d < d_{max}$ and a linearly decreasing weighting function.

Sano and Kita [12, 13] propose to estimate a constant, assuming that all data points follow a normal distribution with variance $\sigma^2 = \sigma_0^2 + k \cdot d$, i.e. the variance is assumed to increase linearly with distance d (cf. Figure 2). They then use maximum likelihood estimators to estimate the constant, σ_0^2 and k. Basically, this is similar, but not equivalent to local regression. The main differences are that the model is global and always takes all samples into account, and that only constants are estimated.

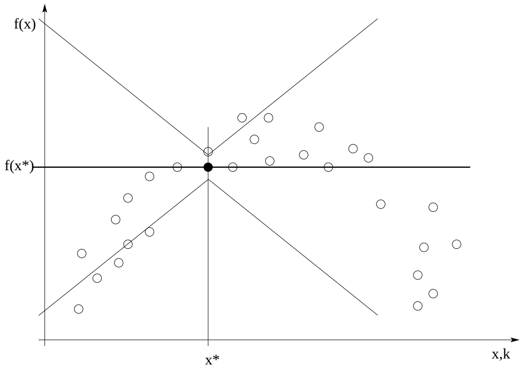


Figure 2: The model of the search space according to [12]. Displayed is the estimated fitness function plus/minus variance.

Table 1: Comparison of different estimation approaches

	New model	Sano/Kita [12]	Branke [4]
Estimation	constant, linear, quadratic	constant	constant
Model	local regression	increasing variance	local regression
Locality	neighborhood self-adaptive, optimized	global	fixed neighborhood
Weighting	sigmoidal weight function	equal for all samples	linear weight function
Variance	locally constant	linear, optimized	locally constant

4 Hill-climber

When linear or quadratic regression is used, as a side product, also the gradient of the fitness landscape is estimated. It is then straightforward to use this information to speed up convergence by hill-climbing.

The hill-climber we use here simply tries to move the individual in the direction of the gradient. The new position x' is accepted with probability $w_h(d)$ only if the estimate $\hat{f}(x') = \hat{g}_{x^*,h}(x')$ is less than the estimate at the old position $\hat{f}(x^*) = \hat{g}_{x^*,h}(x^*)$. If after 10 trials no such location has been found, the individual is not changed.

5 Empirical Evaluation

5.1 Test Functions and Parameter Settings

To obtain some first results, we have selected a simple test function similar to the one used in [12]. Test function f_1 is a simple 5-dimensional parabola with a constant noise, i.e.

$$\begin{aligned} \min f_1(x) &= \sum_{i=1}^{5} x_i^2 + \sigma \text{ with } \sigma \sim N(0,1) \\ x &\in [-2 \ldots 2] \end{aligned}$$

The standard settings for the EA applied in the experiments were real-valued, direct encoding of the decision variables, generational reproduction with elitism of one individual, ranking selection, two-point-crossover with probability 0.8, mutation rate of 0.2, Gaussian mutation with standard deviation of 0.5, and a population size of 50 individuals. Each reported result is the average of 20 runs with different random seeds. The algorithmic variants compared in the next section only differ in the way of estimating the fitness values.

All samples ever generated are kept in the memory and are used for fitness estimation.

5.2 Results

We considered three different performance measures:

1. The true fitness value of the perceived best individual, T^*. This basically corresponds to the fitness one would get when implementing the best individual at any point in time.
2. The true fitness of the actually best individual in the population, B^*. If it is possible to examine the final population more closely, and to actually pick the truly best individual, then this measure would be appropriate.
3. The average deviation of the estimated fitness values from the true fitness value D (cf. [12]).

The three measures are computed as follows:

$$\begin{aligned} T^* &= f(x^t)|x^t : \hat{f}(x^t) \le \hat{f}(x_i), \quad i = 1 \ldots P \\ B^* &= f(x^b)|x^b : f(x^b) \le f(x_i), \quad i = 1 \ldots P \\ D &= \frac{1}{P} \sum_{i=1}^{P} |\hat{f}(x_i) - f(x_i)| \end{aligned}$$

with P being the number of individuals in the current population.

For comparison, in Fig. 3 the performance for an EA using the average of 10 or 100 independent samples for every evaluation is displayed. Note that if number of samples is the determining factor for the computation time, the latter two require 10 or 100 times as many computation time as the other two.

5.2.1 Standard Quadratic Regression

First, we compare the approach described above using quadratic regression with the model used by Sano and Kita [12].

As can be seen, the regression estimators have a significantly lower estimation error than Sano/Kita (Fig. 4 (c)), actually, the estimation error D is in the range of an EA performing 50 independent samples for each evaluation. Considering the true fitness of the perceived best individual, or the true fitness of the actual

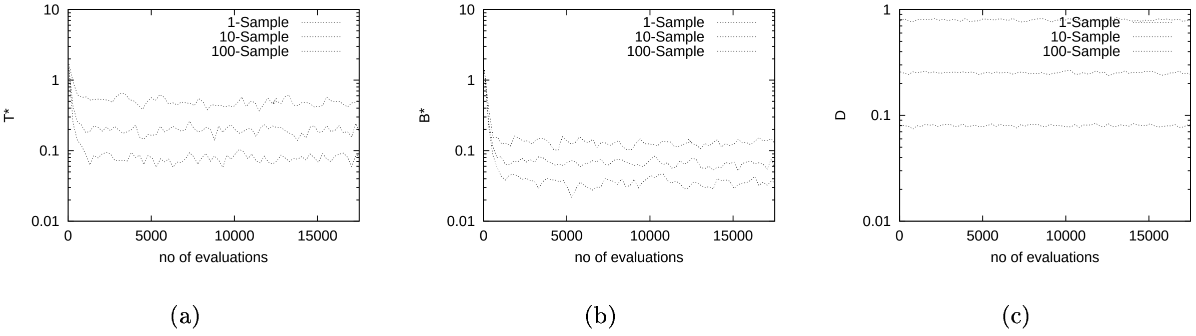


Figure 3: Comparison of the true fitness of the perceived best individual (a), the true fitness of the best individual in the population, and (c) the average estimation error for an EA using 1, 10, or 100 independent samples per evaluation.

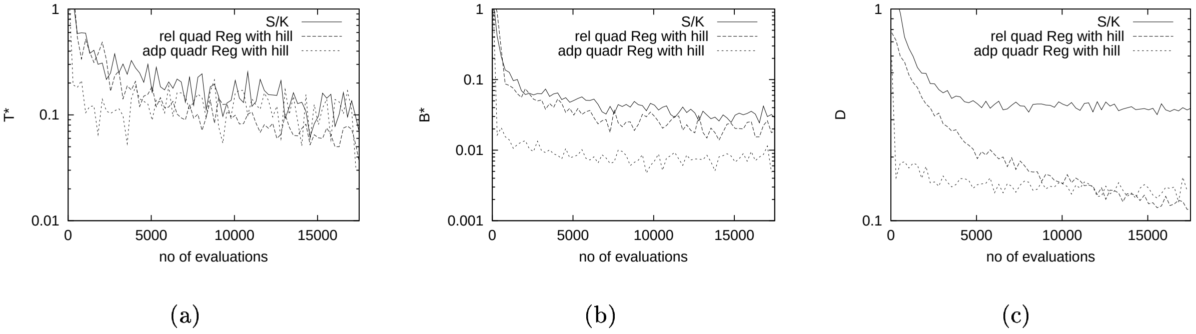


Figure 4: Comparison of the true fitness of the perceived best individual (a), the true fitness of the best individual in the population, and (c) the average estimation error for the Sano/Kita model, quadratic regression with relative neighborhood size and quadratic regression with adaptive neighborhood size.

best individual in the population however, the three compared estimators perform similar, with the adaptive regression model converging faster in the beginning, and the relative regression model displaying a slight advantage towards the end.

5.2.2 Simpler Regression Models

Using simpler regression models (constant, linear) speeds up computation. For the quadratic test function used for testing, naturally quadratic regression would be ideal. In this section we examine by how much the results change when simpler regression models are used.

As expected the higher the regression degree the better the results. The effect increases from estimated best over overall best to deviation.

5.2.3 The Effect of Hill-climbing

Now we examine the effect of using the automatically generated gradient information to integrate a simple hill climber.

As can be seen in Fig. 6, hill-climbing improves the performance for the regression model and all performance measures. The effect on the deviation is negligible, as the model is the same.

6 Conclusion and Future Work

In this paper, we reexamined the idea of fitness estimation taking into account available fitness evaluation from neighboring individuals. We have shown that this idea can be realized on a sound statistical basis by using local regression.

As first experiments on a simple test function have shown, local regression is significantly better in estimation than previous approaches. The estimation error is about equivalent to using 50 independent samples to estimate each individual.

The regression with adaptive neighborhood also produces the best results considering the fitness of the true best individual in the population, and it converges significantly faster than previous approaches.

As a side product, local regression estimates the gradient of the fitness landscape. As we have shown, this information can additionally be used to speed up convergence by integrating a simple local hill-climber. In our experiments, integrating the local hill-climber additionally improved performance by a magnitude.

There still remain opportunities for future work. First of all, a proper setting of the neighborhood parameter h should be further investigated. Then, it should be tested how the reported results transfer to more complicated test functions and also to real world problems like e.g. scheduling. Also, if there is enough computing power available to run many evaluations, the number of samples stored in the memory has to be limited somehow.

References

[1] A. N. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, pages 97–122, 1994.

[2] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence: A Dynamic System Perspective*, chapter 11, pages 152–163. 1995.

[3] D. Arnold. Evolution strategies in noisy environments - a survey of existing work. In *Second EvoNet Summer School on Theoretical Aspects of Evolutionary Ccomputing*. Springer, 2000.

[4] J. Branke. Creating robust solutions by means of an evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, number 1498 in LNCS. Springer, 1998.

[5] J. Branke. Efficient evolutionary algorithms for searching robust solutions. In I.C. Parmee, editor, *Adaptive Computing in Design and Manufacture (ACDM 2000)*, pages 275–286. Springer, 2000.

[6] J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.

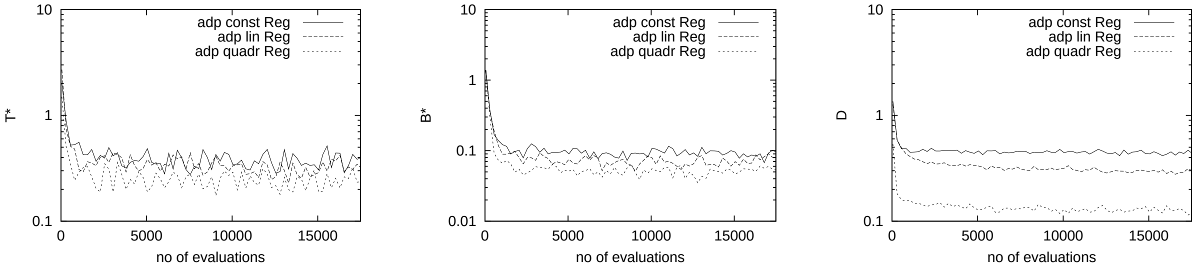
[7] U. Hammel and T. Bäck. Evolution strategies on noisy functions, how to improve convergence properties. In Y. Davidor, H. P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature*, number 866 in LNCS. Springer, 1994.

[8] C. Loader. *Local Regression and Likelihood*. Springer, 1999.

[9] Brad L. Miller. *Noise, Sampling, and Efficient Genetic Algorithms*. PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1997. available as TR 97001.

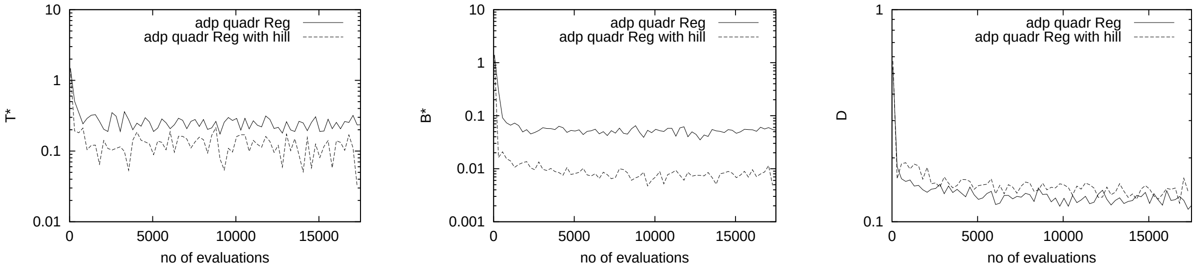
[10] I. C. Parmee. The maintenance of search diversity for effective design space decomposition using cluster-oriented genetic algorithms (COGAs) and multi-agent strategies (GAANT). In *Proceedings of ACEDC'96*, 1996.

[11] C. R. Reeves. A genetic algorithm approach to stochastic flowshop sequencing. In *Proceedings of the IEE Colloquium on Genetic Algorithms for Control and Systems Engineering*, number 1992/106 in Digest, pages 13/1–13/4. IEE, London, 1992.



(a) (b) (c)

Figure 5: Comparison of the true fitness of the perceived best individual (a), the true fitness of the best individual in the population, and (c) the average estimation error for the adaptive regression model, with constant, linear and quadratic regression function.



(a) (b) (c)

Figure 6: Comparison of the true fitness of the perceived best individual (a), the true fitness of the best individual in the population, and (c) the average estimation error for quadratic regression with adaptive neighborhood size, with and without hill-climber.

[12] Y. Sano and H. Kita. Optimizatin of noisy fitness functions by means of genetic algorithms using history of search. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN VI)*, volume 1917 of *LNCS*, pages 571–580. Springer, 2000.

[13] Y. Sano, H. Kita, I. Kamihira, and M. Yamaguchi. Online optimization of an engine controller by means of a genetic algorithm using history of search. In *SEAL*. Springer, 2000. to appear.

[14] P. Stagge. Averaging efficiently in the presence of noise. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature V*, volume 1498 of *LNCS*, pages 188–197. Springer, 1998.

[15] S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–208, 1997.

[16] D. Wiesmann, U. Hammel, and T. Bäck. Robust design of multilayer optical coatings by means of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2(4):162–167, 1998.

Evolving Populations of Expert Neural Networks

Joseph Bruce and Risto Miikkulainen
{jbruce|risto}@cs.utexas.edu
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712 USA

Abstract

In standard neuroevolution, the goal is to evolve one neural network that would compute the right answer most often. However, it often turns out that the population as a whole could perform even better, if we could only choose the right network for each input. One way to do this is to evolve networks that output not only the answer, but also an estimate of that answer's correctness. Experiments in the handwritten character recognition domain show that such an evolutionary process, combined with an effective technique for speciation, can create a population of networks that collectively performs better than any individual network.

1 Introduction

In a typical approach to problem solving with evolutionary methods, a genetic algorithm is used to evolve a population of individuals each attempting to solve the task. The most fit individual found during the evolution, the *champion*, is designated as the final result. For example, when neural networks are evolved for a decision task, the champion is the neural network that is most likely to produce the correct decision for any given input. The rest of the population, and the knowledge and expertise it encodes, is simply thrown away.

However, an analysis of the final population shows that there are often other individuals in the population that are able to produce correct decisions for inputs that the champion cannot handle. Figure 1 shows the fitness of the final population in the handwritten character recognition task. Although the champion only identifies 64% of the characters correctly, 98% of the characters are correctly identified by *at least one* individual in the population.

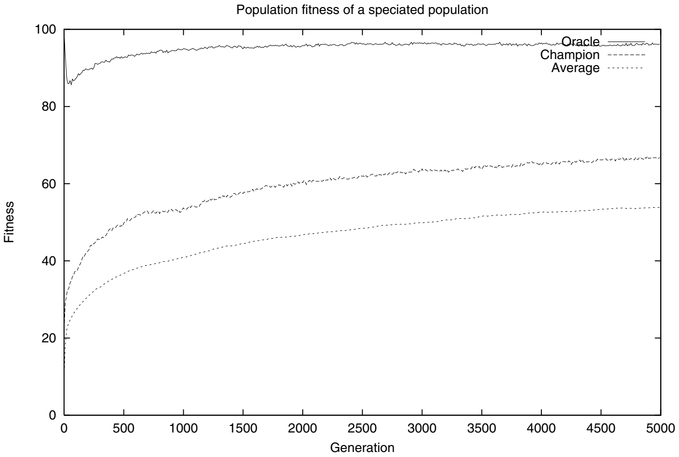


Figure 1: **Percent of correct decisions attainable from a standard neuroevolution population.** Three different measures are used, from top to bottom: the best answer found in the entire population (unrealizable in practice), the answer of the most fit individual, and the population average. Fitness indicates the percentage of correctly-identified test characters in the handwritten character recognition domain (section 4).

We could obtain this level of accuracy by simply choosing the best answer from those produced by the entire population. But how can we determine which individual most likely has the right answer for a given input? One solution is to evolve the individuals not only to produce the answer, but also a level of *confidence* that this answer is correct. The population's answer can be defined as the decision made by the most confident individual.

This idea is tested in the handwritten character recognition task. For the method to work, it is essential to maintain high diversity in the population. Several speciation methods are tested; the island model and its continuous version, the spatial model, are found to be the most effective. With such diversity, the method leads to populations that collectively perform better than any single individual. Confidence evolution of expert neural networks therefore constitutes a promising approach to utilizing the entire population as the result of the evolutionary algorithm.

2 The Method of Confidence Evolution

Figure 1 suggests that in order to make use of the knowledge and expertise in the entire population, the champion might be determined separately for each individual input. Such a selection can be made correctly only based on the correct answer, which is not available during performance. Therefore, although such a high level of performance exists in the population, it is unattainable in practice.

However, it is possible to approximate this selection in various ways. For example, it may be possible to train (or evolve) a meta-level neural network to decide which individual in the population is most likely to produce the right answer for a given input. This is the approach taken for example in the Mixtures of Experts approach [Jacobs et al., 1991], which works well in many supervised tasks. An interesting alternative is to require each individual to rate the quality of each answer it produces. If the population learns to do this effectively, one would be able to outperform the champion by choosing the decision of an individual reporting the highest level of confidence for each decision. This is the approach taken in this paper.

Confidence may be represented by an additional output unit on the neural network. Confidence is that unit's activation when the network is presented with an input. The range of confidence, therefore, is between 0 and 1. To encourage the network to output useful estimates through this unit, the fitness evaluation must be altered. Many fitness evaluations in decision tasks are sums of Booleans, counting the number of correct decisions an individual makes in trials on a training set:

$$f = \sum_i s(\vec{v}_i), \tag{1}$$

where $s(\vec{v}_i) = 1$ if the network's answer on trial i was correct, 0 otherwise.

Training with confidence changes this evaluation in a simple way. It treats each of these trials as a bet. Instead of simply winning 1 each time it is correct, it also stands to lose 1 if it is incorrect. Moreover, the size of the bet is determined by its confidence output $c(\vec{v}_i)$:

$$f = \sum_i s(\vec{v}_i)c(\vec{v}_i), \tag{2}$$

where $s(\vec{v}_i) = 1$ if the answer is correct, $s(\vec{v}_i) = -1$ if it is incorrect. So the network is penalized $s(\vec{v}_i)c(\vec{v}_i)$ for a wrong decision on input $\vec{v}_i$ and is awarded $s(\vec{v}_i)c(\vec{v}_i)$ for a correct decision. This process allows the network to unilaterally set the amount of the bet that its response is correct. It encourages networks to output high confidence only for decisions that are likely to be correct.

The fundamental change to the standard way of evolving neural networks is that in an evolution with confidence, the entire population is considered to be the product of evolution. Answers may be obtained from the population by simply choosing the answer provided by the most confident individual. The specific method for leveraging confidence to extract high-quality decisions is problem-dependent. For example, for some decision tasks the sum of the population's outputs weighted by their confidence might be a useful quantity. But in other domains, such as robotic controllers, it might be better to allow a single individual to provide each decision: weighted sums of different motor outputs could easily lead to a disaster.

In order to use confidence, the population must be diverse enough so that significantly different decisions are made by networks inhabiting distinct niches. A strong method of diversity maintenance (also called speciation or niching), is therefore crucial for success. On the other hand, the restrictions imposed by the speciation technique may adversely affect learning performance for all individuals in the population. A proper speciation technique should strike a balance between these two factors.

3 Methods of Speciation

Speciation is an important design principle in genetic algorithms. Genetic algorithms lose diversity over the course of evolution, converging to a point at which all of the genomes in the population are essentially the same. At this point, crossover operation between two nearly identical genomes is unlikely to create a more fit offspring, and the progress of the GA from that point will be slow, mostly through mutation. A properly speciated population, containing several "niches" of solutions to the task, can continue improving even after some of the niches reach local maxima from which they are unable to improve. It provides for a more reliable search, with many approaches being explored in parallel throughout the evolution.

Speciation techniques are generally implemented in terms of genomes, rather than the structure implemented by the genomes, or that structure's performance, because a diversity of genomes is needed to search the solution space in parallel. However, this diversity of genomes also results in diversity of behaviors. A speciated population contains a wider range of answers—and is more likely to contain at least one correct response for a particular input—than a homogeneous population. Therefore, speciation can be used to create a population where different individuals are responsible for different inputs.

Methods for promoting diversity may involve changes to different parts of the canonical genetic algorithm. In this paper we will compare speciation techniques that modify the GA's selection scheme, the replacement scheme, and the fitness evaluation function. Also crucial to population diversity is the scaling scheme, i.e. the algorithm that converts the individuals' fitnesses into probabilities of being included in the next generation, either unchanged or combined with another genome by crossover. An aggressive scaling scheme that rewards slightly fitter individuals with much higher probabilities will quickly lead to convergence, as genetic material possessed by moderately-fit individuals will be lost in each generation. More subtle scaling schemes are desirable (and also used in this paper) to delay population convergence.

A very simple approach to speciation is to arbitrarily divide the population into non-interacting subpopulations, or islands. A genome cannot perform crossover with any genome of another island, and a newly created individual may replace only a genome in the island of its parents. In some versions, the island model provides for a small rate of migration between islands. Without migration, this approach is equivalent to running a genetic algorithm independently on each of the islands. Even this trivial approach to speciation can be useful; if the genomes in one island reach a plateau early, others may continue improving. This difference is not directly promoted; it is simply allowed to occur by chance. Under migration, populations on an island are allowed time to make small adjustments before competing with outside genomes [Mühlenbein, 1991].

A more general, continuous version of the island method is the spatial, or topological, method. Each individual may inhabit a vertex of an undirected graph, and it may only perform crossover with an individual connected to it through an edge. The resulting offspring may only replace the least fit of its parents, only if the offspring is more fit. This setup is more biologically plausible than the usual "panmictic" populations in which any individual may mate with any other. It prevents premature convergence since a particular genome can spread only to immediate neighbors in a single timestep [Kephart, 1994].

The implementation of a spatial population described above also incorporates the more general speciation strategy called preselection, which stipulates that a newly created individual in a population may only replace one of its parents. This protects against premature convergence because it ensures that at least some of an individual's genetic material will survive into the next generation [Goldberg, 1989].

Fitness sharing is a technique that penalizes genomes that inhabit neighborhoods of many other genomes. Generally, an individual's fitness evaluation is divided by a sharing factor that measures the genome's proximity to others in the population. Genomes in heavily populated peaks receive a high penalty, which translates into a lower probability of propagating to the next generation. This technique is intended to spread the population across several peaks in the solution space, with larger (wider or higher) peaks able to support more individuals. Implicit sharing is a variation in which, for each input, only the individual with the best response from a randomly-selected subset of the population is awarded fitness for that input [Darwen and Yao, 1996].

Crowding is a generalization of preselection, where an individual only replaces a genome to which it is similar (but not necessarily the parent). Under crowding, after a new genome is created, a subset of genomes is randomly selected from the population. The genome in the subset which is closest to the new genome is chosen to be replaced by the new genome [Goldberg, 1989].

The confidence method was tested in conjuction of each of the above speciation methods. Interestingly, their performance was found to differ a lot in the handwritten character recognition domain, which will be described next.

4 Experiments

The method of confidence evolution was applied to the standard benchmark task of recognizing handwritten digits. There are many methods developed specifically for this task. The present goal is not to compete with them, but rather to test the viability of the method and to refine it further. This task is well-suited for such analysis because the correct decisions are readily available. After the method has been tested and refined, it will be possible to apply it to other task where the correct performance is not known.

The data set used was the freely-available subset of 2,992 examples of handwritten digits 0..9 in the NIST database, scaled to an accuracy of 8 × 8 pixels [Choe et al., 1996]. The networks had an input layer of 64 units to encode the 8 × 8 input representing a digit to be classified. The input layer was fully connected to a hidden layer of 20 units, which was fully connected to an output layer of 11 units, representing each of the ten possible digit classifications, and an additional unit to output the confidence in this classification. The output unit with the highest activation was chosen as the classification. The genome represented the real values of the weights and biases of the network. While the canonical GA operates only on binary strings, analogous operations of crossover and mutation were implemented for the real-valued genome: a uniform crossover could take place between weights, and each weight was mutated with a 0.01 probability by adding a normally distributed random value of 0 mean and 1.0 standard deviation to the weight.

The genetic algorithm proceeded on a population of 100 individuals for 5,000 generations. During each generation, the fitness for each network was calculated according to equation 2 (and equation 1 for those experiments where confidence was not used) on a training set of 2,000 patterns, selected randomly among the 2,992 for each experiment. Fitness scaling was done by sigma truncation scaling [Goldberg, 1989], which tolerates negative fitness values. Selection was fitness-proportionate. Throughout evolution, each population was tested on a randomly-chosen test set of 200 patterns not part of the training set.

Notice that a more accurate fitness evaluation could easily be designed, such as the sum of squared errors between the outputs and the target output. Such an evaluation would capture more information about the differences between individuals; however, Boolean values were chosen in order to emulate less well-understood decision tasks for which such detailed information is not available. Furthermore, the fitness evaluation seeks to reward only the desired outcome (the correct classification in the winner-takes-all sense), not any specific way of attaining the outcome. Having such an open-ended fitness evaluation allows the network to implement its own, possibly unexpected, method for achieving the result [Floreano and Urzelai, 2000].

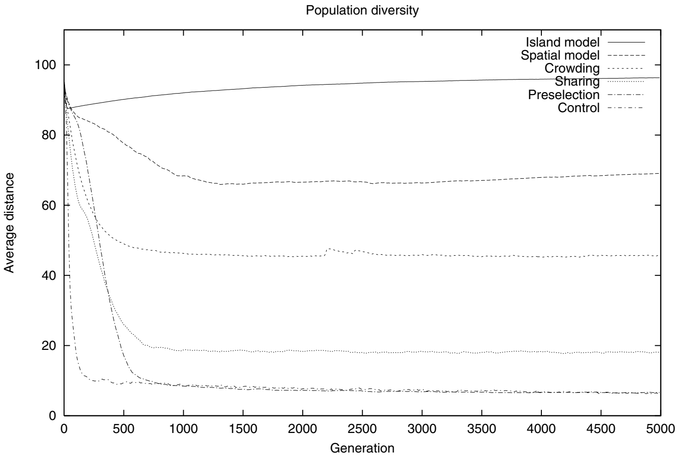


Figure 2: **Population diversity under different speciation methods.** The plots (from top to bottom) show the maximum, average and minimum Euclidean distances between genomes in the population as evolution progresses. Plots are each averaged over 10 independent evolutions.

The different speciation techniques outlined in Section 3 were each tested as part of the confidence evolution method. The island method was implemented by splitting the 100 genomes into 10 noninteracting islands of size 10, with no migration. A spatial population was laid out on a 10 × 10 grid with edges folded back to create a torus; an individual could mate only with one of its four neighbors, with the offspring replacing the parent with lower fitness, only if the offspring's fitness was higher. Fitness sharing was implemented by penalizing genomes according to Euclidean proximity to others in the population. Crowding was implemented such that a new individual was placed in the population in place of the closest Euclidean neighbor in a random subset of 10 from the population. In preselection, an offspring replaced the parent with the lowest fitness if the offspring was more fit than that parent.

5 Results

To varying degrees, each speciation method was able to maintain diversity in the evolution. For each of these methods, the average Euclidean distance between the 100 genomes in the population throughout an evolution without confidence is plotted in figure 2. The island model and the spatial model, i.e. those methods that directly restrict mating to a local neighborhood, resulted in particularly good genomic diversity. A control evolution with no speciation is shown as well. The

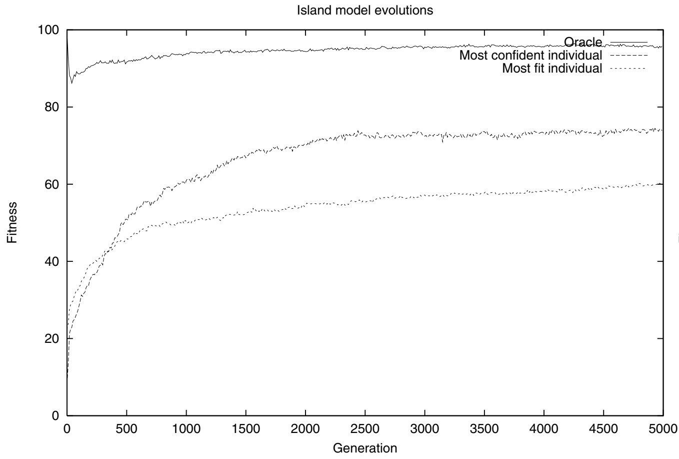


Figure 3: **Accuracy of confidence evolution with islands.** On top, the fitness obtained by choosing (through an unrealizable oracle) the best answer in the entire population is plotted. In the middle, the fitness obtained by choosing the answer of the most confident individual is shown. The fitness of the most fit individual is plotted in the bottom. The plots are averages over 10 runs.

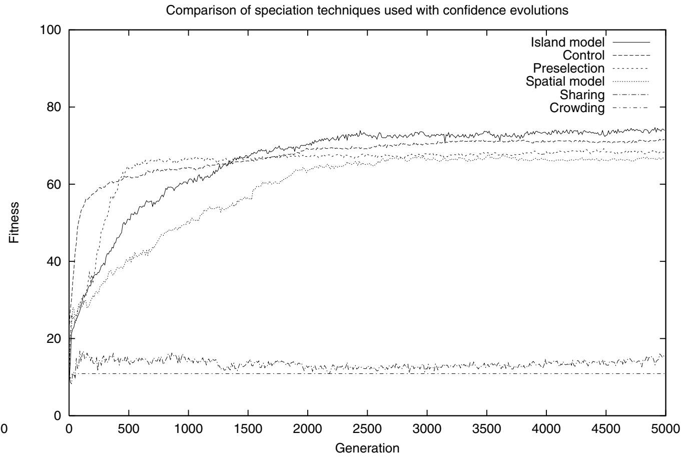


Figure 4: **Accuracy of confidence evolution with different speciation methods vs. standard evolution.** The island method performs the best; the other methods are weaker than the standard evolution, labeled "Control", in the order shown in the legend. The plots are averages over 10 runs.

control evolution quickly lost much genomic diversity, bottoming out at approximately generation 200. This result highlights the need for speciation.

Evolutions with crowding, sharing, and preselection each slowed convergence compared to the evolutions with no speciation, but were considerably less effective. These techniques do not restrict replacement as strongly as the island and the spatial models do; they involve a high degree of randomization in the choice of which population member a new genome should replace. Since this decision is randomized, sampling error can affect replacement, causing genetic drift. Mahfoud (1992) cites this stochastic error as a difficulty with crowding and preselection.

As expected, those speciation methods that maintained the highest diversity also provided the best advantage for confidence evolution. Populations evolved using the island model and the spatial model were diverse enough so that choosing the answer of the most confident individual resulted in better performance than could be obtained from the population's champion. Figure 3 compares the accuracy in the test set of the (unattainable) best answers, answers selected by confidence, and the most fit individual for the island model.

The accuracy of the different methods in the test set is compared in figure 4. The main result is that confidence evolution with islands resulted in a slightly higher level of performance than the control evolution. The other methods were worse, underlining the importance of an effective speciation method in confidence evolution. Crowding, in particular, did not perform above chance; apparently one or more high-bidding and wrong individuals persisted thoughout the evolution, never being replaced because they were too far from the other genomes. This result demonstrates the difference between random diversity (such as the population before the first generation), and useful diversity (niches of high-performing but different individuals). Only the latter kind of diversity is useful for confidence evolution.

Interestingly, when speciation methods were added to the standard evolution, the performance either was not affected or actually became worse. This result suggests that making full use of speciation requires a technique such as confidence. It also shows that the observed improvement over standard evolution is indeed due to confidence, and not speciation.

6 Discussion and Future Work

The experimental results in this paper show that confidence evolution can improve performance of neuroevolution in the handwritten digit recognition task. They also show that effective speciation is crucial for this technique. How general are these results?

Speciation is generally used in a genetic algorithm to increase the overall rate of learning by slowing down population convergence. However, when speciation is used in conjunction with confidence evolution, the goal is instead to increase the variation in answers made. The existing speciation techniques used in this paper may not be the best for this new, slightly different goal. This issue is underscored by the fact that confidence provided the greatest advantage in evolutions with completely noninteracting subpopulations – a technique that would tend to harm overall fitness since small populations are being evolved in each island, leading to cruder solutions.

In the extreme, a speciation technique that even significantly decreases the overall fitness of the population would work well with confidence evolution if it maintains a large variety of correct answers. This is a tradeoff that is not acceptable in a standard genetic algorithm: if the goal of speciation is to increase the overall rate of learning, a technique that lowered the fitness of all individuals significantly would not be useful. But given this new set of criteria, perhaps such strong speciation techniques could be devised in the future, gaining even more benefit from confidence evolution than is possible with the current techniques [Ackley, 1987, Katila, 1987].

It is also possible that domains other than character recognition might be more amenable to the current speciation techniques. This domain benefits from a large number of individuals fine-tuning an approximate solution in a small space, which requires exactly the convergence that speciation attempts to avoid. Instead, genetic algorithms in general and speciation methods in particular are strongest at quickly finding approximate solutions. Therefore, problems that involve more global search may be more amenable to the current techniques.

In particular, genetic algorithms are the method of choice for sequential decision tasks where the correct answers are not known and the feedback is highly sporadic [Moriarty and Miikkulainen, 1997, Moriarty et al., 1999]. Given the promising results in the handwritten character recognition domain, confidence evolution should work well in such tasks. For example, imagine applying confidence to the training of a robotic controller. Each neural controller in the population would be presented with an encoding of the robot's sensory input, and it would output a motor action and a confidence level. The action recommended by the most highly confident controller would be selected. After several decisions were made, a fitness evaluation for the whole sequence of decisions would be obtained. This fitness would then be distributed to the controllers according to how confident they were of their outputs and how often they were selected.

Although at first it seems that such fitness information might be too noisy, the situation is very similar to those of SANE and ESP neuroevolution methods described in [Moriarty and Miikkulainen, 1997], [Moriarty, 1997], [Gomez and Miikkulainen, 1997], and [Gomez and Miikkulainen, 1999], where populations of neurons are evolved to form good neural networks. Each neuron receives a fitness based on how well the whole neural network performed in the task: in effect, the neurons are evolved to speciate into useful subtask that work well together. In reinforcement learning tasks with confidence evolution, similarly each network is rewarded based on how well the whole population did in the sequence of decisions. Given how powerful the SANE and ESP methods are, this same approach may also work well in confidence evolution.

In other learning tasks, it may be useful for an agent to express its confidence in a more direct form, by answering a more specific question about its performance. For example, a neural robotic controller might estimate the amount of time needed to reach a goal state, rather than estimating its probability of success. Such fitness functions might lead to more powerful evolution. Similarly, instead of always selecting the most confident individual's answer, the answer might be constructed by combining answers of the most fit individuals, or by at least not considering the answers of those with the lowest fitness. This method would for example solve the problem that occurred with crowding in the above experiments.

Different techniques of training individuals to output confidence could also be considered. Evolving networks to provide answers and confidence estimates is clearly a more difficult task than simply evolving networks to provide answers. Might this increased difficulty be offset by using a combination of evolution and learning, or by Lamarkian evolution? Might it be beneficial to evolve the confidence neuron or network in a separate phase of evolution? Or perhaps as a separate network entirely? These are some of the issues that will be explored in future work.

7 Conclusion

This paper shows how the knowledge and expertise encoded by the entire evolved population can be utilized to obtain a high level of performance. High-quality decisions may be extracted from the population if a fitness evaluation rewards individuals that accurately output estimates of the quality of their decisions. To use this technique, a diverse population, capable of producing many different correct answers, is needed. This research motivates the development of techniques to ensure a high level of diversity throughout evolution, possibly even at the expense of overall fitness. The technique of confidence may have broad applicability in the domain of reinforcement learning tasks, which is the main direction of future work.

References

[Ackley, 1987] Ackley, D. (1987). A Connectionist Machine for Genetic Hillclimbing. Boston: Kluwer.

[Choe et al., 1996] Choe, Y., Sirosh, J., and Miikkulainen, R. (1996). Laterally interconnected self-organizing maps in hand-written digit recognition. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 736–742. Cambridge, MA: MIT Press.

[Darwen and Yao, 1996] Darwen, P. and Yao, X. (1996). Every niching method has its niche: Fitness sharing and implicit sharing compared. *Parallel Problem Solving from Nature*, pages pp. 398–407.

[Floreano and Urzelai, 2000] Floreano, D. and Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13:431–443.

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

[Gomez and Miikkulainen, 1997] Gomez, F. and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342.

[Gomez and Miikkulainen, 1999] Gomez, F. and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Denver, CO. Morgan Kaufmann.

[Jacobs et al., 1991] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3:79–87.

[Katila, 1987] Katila, R. (2000). In Search of Innovation: Search Determinants of New Product Introductions. Doctoral Dissertation, the University of Texas at Austin.

[Kephart, 1994] Kephart, J. O. (1994). How topology affects population dynamics. In Langton, C. G., editor, *Artificial Life III*, volume XVII. Addison-Wesley, Reading, MA.

[Mahfoud, 1992] Mahfoud, S. (1992). Crowding and preselection revisited. *Parallel Problem Solving from Nature*, 2:27–36.

[Moriarty, 1997] Moriarty, D. E. (1997). *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin. Technical Report UT-AI97-257.

[Moriarty and Miikkulainen, 1997] Moriarty, D. E. and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive co-evolution. *Evolutionary Computation*, 5:373–399.

[Moriarty et al., 1999] Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229.

[Mühlenbein, 1991] Mühlenbein, H. (1991). Evolution in time and space - the parallel genetic algorithm. In Rawlins, G., editor, *Foundations of Genetic Algorithms*. Morgan-Kaufmann.

A Hybrid Genetic Search for the Sorting Network Problem with Evolving Parallel Layers

Sung-Soon Choi and **Byung-Ro Moon**
School of Computer Science and Engineering,
Seoul National University,
Seoul, Korea
{irranum,moon}@soar.snu.ac.kr

Abstract

In this paper, we propose a hybrid genetic algorithm for the optimal sorting network problem. Based on a graph-theoretical viewpoint, we have devised a new local optimization heuristic. We also propose a new encoding scheme and have devised a new crossover and mutation operator. Using a single-CPU PC, we obtained results comparable with previous results obtained using supercomputers.

1 Introduction

A sorting network is a hardware logic in which inputs are sorted through a number of homogeneous comparators. We denote a comparator by c(a,b). In a comparator c(a,b), the a^{th} bus and b^{th} bus are compared; if their values are in order, they are ignored, otherwise they are exchanged. Figure 1 shows a simple sorting network [c(0,1), c(2,3), c(0,2), c(1,3), c(1,2)]. This is a 4-bus sorting network with five comparators. Here, each vertical edge represents a comparator that connects two buses. The four inputs in the buses 0, 1, 2, 3 are sorted after passing the five comparators.

We can usually run a number of comparators simultaneously. In Figure 1, the first and second comparators can run simultaneously, since they are independent. Similarly, the third and the fourth comparators also can run simultaneously. Thus, the sorting can be completed in just three parallel steps.

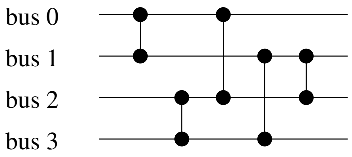


Figure 1: A 4-bus sorting network

Sorting networks have been studied both in the academic world and in industry. The research has two main aims [14]. One is to reduce the number of comparators and the other is to reduce the number of parallel steps to minimize the running time. We focus on minimizing the number of comparators following the convention [11] [7] [13]. We focus especially on the 16-bus case.

Historically many scientists had made a study of 16-bus sorting networks [3] [9] [1] [15] [10]. In 1969, Green [10] first discovered a 60-comparator sorting network which is still one of the best known. Recently, 16-bus sorting networks have attracted attention again due to the improvements in new stochastic search methods [11] [2] [7] [13] [8].

In 1992, Hillis [11] tackled this problem by a genetic algorithm on a CM-1 supercomputer. First, he initialized each network with the first 32 comparators of Green's network so that the size of the problem space was reduced, and then evolved the networks by a genetic algorithm with co-evolution. He found a 61-comparator sorting network.

Although he failed to reach the quality of the best known, his record was encouraging as the first trial with a genetic algorithm. After that, Drescher [7] attacked the same problem as Hillis and found 60-comparator sorting networks, whose quality matches the best known. He also used a genetic algorithm on a CM-5 supercomputer. In 1995, Juillé [13] also found 60-comparator sorting networks with a stochastic search method called END (Evolving Non-Determinism), without fixing the first 32 comparators, on a Maspar MP-2 supercomputer.

Because of the huge computational needs, all these studies used supercomputers. In [6], we used a single-CPU Pentium PC and found 60-comparator sorting networks with a 76% probability in average 30 minutes. In this study, we propose a way to improve this result. This is thought to be possible by devising effective local optimization heuristics and combining them with genetic operators.

The rest of this paper is organized as follows. In Section 2, we present our theoretical view of the sorting network problem. In Section 3, we describe our local optimization heuristics and in Section 4 we provide a genetic algorithm combined with the local optimization heuristic. We give the experimental results in Section 5. Finally, we make our conclusions in Section 6.

2 Preliminaries

2.1 Problem Definition

To check the validity of an n-bus network, we have to be assured that the network correctly sorts all possible input sequences. There are $n!$ sequences and it is costly, sometimes intractable, to check all of them. Fortunately, the Zero-One Principle [14] says that we can prove the validity of the network by testing just 2^n binary sequences instead of $n!$ sequences.

We denote by $\mathcal{T}_n$ the entire set of 2^n binary sequences with which we test and by $\mathcal{S}_n$ the set of sorted sequences from $\mathcal{T}_n$. ($\mathcal{S}_n \subseteq \mathcal{T}_n$) For example, $\mathcal{T}_4 = \{0000, 0001, 0010, \ldots, 1111\}$ and, consequently, $\mathcal{S}_4 = \{0000, 0001, 0011, 0111, 1111\}$. We can consider an n-bus sorting network as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Let f_χ be the function corresponding to a sorting network χ. Then the entire set of valid sorting networks, $\mathcal{V}_n$, can be defined as follows:

$$\mathcal{V}_n = \{\chi \mid f_\chi(\mathcal{T}_n) = \mathcal{S}_n\}.$$

If we denote by $|\chi|$ the number of comparators in the sorting network χ, then the n-bus minimum sorting network problem is to find $\chi^* \in \mathcal{V}_n$ such that $|\chi^*| = \min\{|\chi| \mid \chi \in \mathcal{V}_n\}$. If we fix the comparators in the front part of networks in advance, we can reduce the problem search space considerably; this is because we only have to test the sequences not sorted by the fixed comparators out of the 2^n binary sequences. Let ϕ be a sequence of comparators. The set of binary sequences not sorted after passing ϕ is $f_\phi(\mathcal{T}_n) - \mathcal{S}_n$. Then, in the case that fixes the comparators of ϕ, the minimum sorting network problem is finding ψ^* such that

$$|\psi^*| = \min\{|\psi| \mid f_\psi(f_\phi(\mathcal{T}_n) - \mathcal{S}_n) \subseteq \mathcal{S}_n\}.$$

In the 16-bus sorting network problem, we fixed the sequence ϕ_{32} of the first 32 comparators following Hillis [11]. (i.e., $|\phi_{32}| = 32$) Then we only have to consider the 151 sequences of $f_{\phi_{32}}(\mathcal{T}_{16}) - \mathcal{S}_{16}$ among all the 2^{16} binary sequences of $\mathcal{T}_{16}$.

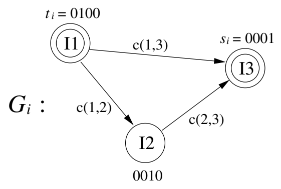


Figure 2: Problem space graph for $t_i = 0100$

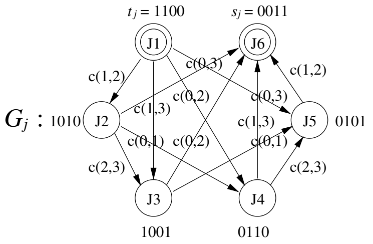


Figure 3: Problem space graph for $t_j = 1100$

2.2 Graph Representation for Problem Spaces

For an element $t_i \in \mathcal{T}_n$, let s_i be the sequence produced by sorting t_i. ($s_i \in \mathcal{S}_n$) For example, in 4-bus networks, if $t_i = 0100$, then $s_i = 0001$. Now we consider a directed graph G_i, which has t_i as the starting point and s_i as the end point. (See Figure 2.) Here, c(x,y) represents the comparator that connects input buses x and y. Each vertex represents the state of the binary sequence after sorting t_i by the comparators indicated by the edges from the starting point to the vertex. In the above graph, the minimum network for sorting t_i is the network composed of only c(1,3), which is the only comparator on the shortest path from the starting point to the end point.

We now consider $t_j = 1100$ and, consequently, $s_j = 0011$ in a similar way. Then we have a graph G_j as in Figure 3. In this case, there are four distinct minimum sorting networks with two comparators that sort t_j: [c(1,2), c(0,3)], [c(1,3), c(0,2)], [c(0,2), c(1,3)], and [c(0,3), c(1,2)].

If the test set is composed of only one sequence as in Figure 2, the problem of finding a valid minimum sorting network is just a shortest path problem in the graph corresponding to that sequence. However, it is not trivial when the test set has more than one sequence. For instance, it is not trivial to find a minimum network that sorts both t_i and t_j by individually looking at G_i and G_j in the above example. We combine the two graphs G_i and G_j for this. We denote by $G_i \bigotimes G_j$ the graph obtained after combining G_i and G_j. Figure 4 shows $G_i \bigotimes G_j$.

Then, from $G_i \bigotimes G_j$, it is easy to find the minimum sorting network for the two test sequences. In this

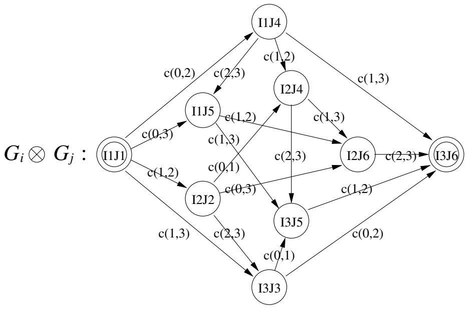


Figure 4: Combined problem space graph

case, there are two distinct minimum networks with two comparators: [c(0,2), c(1,3)] and [c(1,3), c(0,2)].

In this way, we can represent the problem space of an n-bus sorting network as a graph. First, we construct a graph G_i for each $t_i \in \mathcal{T}_n$. Then, we combine all these graphs to get the graph representing the problem space of n-bus networks. In the case where a set ϕ of comparators is fixed, we only have to combine the graphs corresponding to the sequences in $f_\phi(\mathcal{T}_n) - \mathcal{S}_n$. That is, the problem space graph becomes

$$G = \bigotimes_{i=1}^{|f_\phi(\mathcal{T}_n)-\mathcal{S}_n|} G_i \ .$$

In the end, the problem of finding a minimum sorting network is equivalent to finding a sequence of comparators obtained by solving a shortest path problem from the starting point to the end point of G. When the input size is small, it is not difficult to construct such a graph G. However, the size of the graph sharply increases in relation to the input size. Even when we fix the set ϕ_{32} of the first 32-comparators in the 16-bus network problem, it is impossible to construct the problem space graph with a tractable space [6].

2.3 O-Pairs and N-Pairs

Let $t(i)$ be the i^{th} bit value of a binary sequence t. For two input bus indices x and y such that $x < y$ $(x, y = 0, 1, 2, ..., n-1)$, if $(f_\chi(t))(x) \leq (f_\chi(t))(y)$ for all $t \in \mathcal{T}_n$, then the sorting is acceptable with the sorting network χ as far as the two input buses are concerned. In this case, we call such an input bus pair (x,y) an ordered pair (o-pair) with respect to the network χ. On the other hand, if there exists a $t \in \mathcal{T}_n$ such that $(f_\chi(t))(x) > (f_\chi(t))(y)$, then the sorting is not guaranteed for the two buses. In this case, we call such an input bus pair (x,y) a non-ordered pair (n-pair) with respect to the network χ.

Figure 5 shows an example of 4-bus networks. Whatever sequences we provide for this network, the input bus pair (0,1) is eventually in order. The input bus pairs (0,2) and (0,3) are also in order. That is, input bus pairs (0,1), (0,2) and (0,3) are o-pairs for the above network. On the other hand, for the input bus pair (1,2), there exist unordered input sequences such as $t_i = 0100$. Thus (1,2) is an n-pair. The pairs (1,3) and (2,3) are also n-pairs.

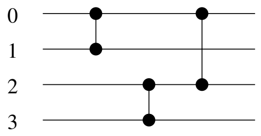


Figure 5: An invalid network

We denote by $OP(\chi)$ the entire set of o-pairs for a network χ and by $NP(\chi)$ the entire set of n-pairs for it. Similarly, we use $OP(v)$ and $NP(v)$ for the vertex v corresponding to the network χ in the problem space graph. It is clear that $|OP(\chi)| + |NP(\chi)| = \binom{n}{2}$.

2.4 Parallel Layers

In a sorting network, a number of consecutively located independent comparators are allowed to be shuffled. Figure 6 shows an example. In the figure, the first and the second comparators can be interchanged. Similarly, it is also possible to interchange the third and the fourth. We handle these interchangeable comparators as a set, since the sequence of these comparators does not affect the function of the network. We call such a set of comparators a parallel layer. The network in Figure 6 is composed of three parallel layers.

A parallel layer strongly affects the subsequent search direction. In the 16-bus sorting network problem, the four parallel layers, corresponding to the set ϕ_{32} of the first 32-comparators, are generally fixed in the front part of a network. Of course, fixing these four parallel layers significantly narrows down the search space.

If a considerable number of leading parallel layers have been determined, the rest is straightforward. For example, if the first 55 comparators of a 60-comparator sorting network have been determined, it is fairly easy to find the remaining five comparators by the local optimization heuristic described in the next section. That

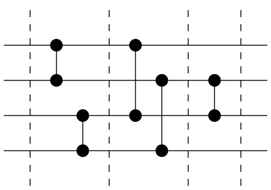


Figure 6: An example of parallel layers

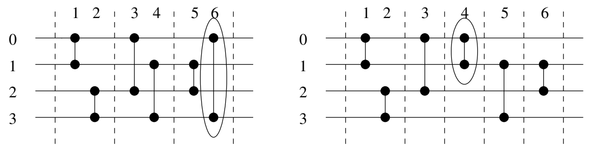


Figure 7: Examples of redundant comparators

is, from the perspective of the parallel layers, the sorting network problem can be considered to be the problem of finding a considerable number of leading parallel layers. For this reason, we evolve only a fixed number of parallel layers from the front. The details are described in section 3 and section 4.

3 Local Optimization

The local optimization consists of two sequential subprocesses: edit and repair. It is performed after crossover and mutation in our genetic algorithm.

3.1 Edit

The edit removes redundant comparators in a network. Consider the networks in Figure 7. Comparators are indexed 1 through 6 for the convenience of explanation. Both networks represent valid networks. However, in the first network, the input bus pair (0,3) is an o-pair with respect to the comparators 1 through 5; comparator 6 is redundant. The network is still valid after removing comparator 6 from the network. Redundant comparators do not necessarily exist only in the last part of a network. In the second network, the input bus pair (0,1) is an o-pair with respect to the comparators 1, 2, and 3; comparator 4 is redundant.

As shown in the above example, if an input bus pair corresponding to a comparator is an o-pair with respect to its previous comparators, the comparator is redundant. We can find all the redundant comparators as follows. First, we provide all the test sequences to the network and perform sorting. In this process, we check the comparators that do not exchange any input pair. The input bus pair corresponding to such a comparator is an o-pair even without it. All these redundant comparators are removed in the edit process. After the edit process, there may be some parallel layer that has no comparator. In the second network of Figure 7, for example, there is no comparator in the third parallel layer after the edit process. In this case, we shift ahead the subsequent parallel layers.

3.2 Repair

An offspring produced as a result of crossover and mutation is usually not valid. The repair process modifies an invalid network to a valid one by adding new comparators.

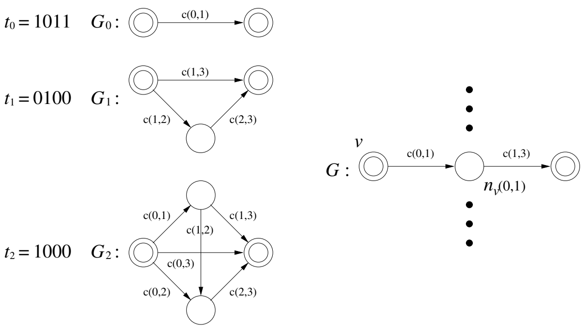


Figure 8: An example of related sequences

3.2.1 Appending

As seen in the previous section, the problem space can be represented by a graph G. Here, every vertex except the end point corresponds to an invalid network. The process of appending a new comparator to the network, in the repair process, corresponds to a move between two adjacent vertices in the graph. In this process, a shortest path means a minimal use of comparators.

However, the size of the problem space is usually too large and, consequently, it is not generally possible to deterministically find a shortest path from a given vertex (an invalid network) to the end point in tractable time. In the end, we cannot help but reach the end point by searching only a limited number of neighborhood vertices for each vertex and selecting a promising one among them.

We call the adjacent vertices of a vertex v in a graph G the explicit neighbors of v. We denote by $\mathcal{EN}(v)$ the set of the explicit neighbors of v. We choose a vertex from $\mathcal{EN}(v)$ that reduces the most number of n-pairs. It turns out through experiments that the number of n-pairs is a fairly useful measure in selecting a promising neighbor.

Figure 8 shows an example. For a 4-bus invalid network χ', assume that there are three sequences t_0, t_1, and t_2 in $f_{\chi'}(\mathcal{T}_4) - \mathcal{S}_4$. Let $n_v(i,j)$ be the explicit neighbor vertex in G to which v is transited by comparator c(i,j) among the elements of $\mathcal{EN}(v)$. ($n_v(i,j) \in \mathcal{EN}(v)$) In this example, if we sort t_0 and t_1, then t_2 is sorted accordingly. By measuring the number of remaining n-pairs, we would select the neighbor $n_v(0,1)$. Table 1 shows the number of remaining n-pairs after transition by each possible comparator.

In a graph G, let $d_G(v)$ be the shortest distance from vertex v to the end point. Then, we can get the following bound for $d_G(v)$ using the number of n-pairs

Table 1: The number of n-pairs for each $n_v(i, j)$

$n_v(i,j)$	$n_v(0,1)$	$n_v(0,2)$	$n_v(0,3)$	$n_v(1,2)$	$n_v(1,3)$
1011 0100 1000	0111 0100 0100	1011 0100 0010	1011 0100 0001	1011 0010 1000	1011 0001 1000
$\lvert NP(n_v(i,j))\rvert$	2	4	3	4	3

[6].

Fact 1 *For* $G = \bigotimes_i G_i$,

$$\max\{d_{G_i}(v)\} \leq d_G(v) \leq |NP(v)| \ .$$

3.2.2 Insertion

Not only do the comparators that are "appended" to a network transit the vertex v to promising neighbors. The comparators "inserted" in the middle of the network also have the potential to transit v to promising neighbors. When we consider an invalid network χ' as a sequence of comparators, there are $|\chi'| + 1$ candidate positions in which we can add a new comparator in the repair process.

When we insert (as opposed to "appending") a comparator in the middle of χ', the comparator does not necessarily transit v to an adjacent vertex of v in the graph G. We call such a vertex, which is not adjacent to v but can be transited to by a comparator insertion, an implicit neighbor of v. (Figure 9) We denote by $\mathcal{IN}(v)$ the set of the implicit neighbors of v.

In selecting a promising neighbor of v, we are able to search the problem space more extensively by considering $\mathcal{IN}(v)$ as well as $\mathcal{EN}(v)$. However, it is too expensive to consider all the elements of $\mathcal{IN}(v)$. We restrict the search by protecting the boundaries of parallel layers.

If we allow a comparator to be inserted in any position of a network, the structure of the parallel layers that an offspring inherits from the parents can be distorted considerably. Since the repair process is performed after crossover and mutation, the repair with this type of insertion may often prevent an offspring from inheriting the structures of the parallel layers in the parents.

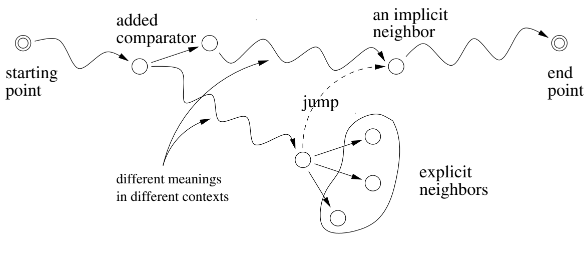


Figure 9: A symbolic representation of an implicit neighbor

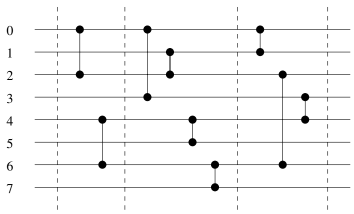


Figure 10: An example of 8-bus network

In this context, we consider only the comparators that do not break the structure of the parallel layers in the repair process. Figure 10 shows an example 8-bus network. In the figure, the buses 0, 2, 4, and 6 have already been used in the first parallel layer. So, we do not consider comparators that connect at least one of these buses in the first layer. Instead, we consider only the $\binom{4}{2} = 6$ comparators that connect two unused buses. In the second layer, we do not consider any comparators since all the buses in the layer have been used. In the third layer, there are two unused buses and we thus consider only the comparator c(5,7).

In this manner, we can search the elements of $\mathcal{IN}(v)$ reasonably maintaining the structure of parallel layers. We call the vertices, which are not adjacent to v but to which v is transited in this way, the restricted implicit neighbors of v. We denote by $\mathcal{RN}(v)$ the set of the restricted implicit neighbors of v. It is clear that $\mathcal{RN}(v) \subseteq \mathcal{IN}(v)$.

We consider the elements of $\mathcal{RN}(v)$ with respect to the fixed number of parallel layers and the elements of $\mathcal{EN}(v)$ in the repair process. It seems to be desirable to select a comparator that reduces the number of n-pairs as much as possible among the comparators that transit v to $\mathcal{RN}(v) \cup \mathcal{EN}(v)$. Here, an inserted comparator can make new redundant comparators after it, unlike the appended comparators. We remove such redundant comparators from the network using the edit process; the length of the network can be even smaller than the network before the insertion. Thus, we also consider the edited lengths of the networks when we evaluate the elements of $\mathcal{RN}(v)$.

We select the promising neighbor of v in the following priority:

1. A vertex in $\mathcal{RN}(v)$ such that its corresponding network has fewer comparators than the network of v and the number of its n-pairs is also smaller than $|NP(v)|$.
2. A vertex in $\mathcal{RN}(v)$ such that its corresponding network has fewer comparators than the network of v and the number of its n-pairs is the same as $|NP(v)|$.
3. A vertex in $\mathcal{RN}(v)$ such that its corresponding

```
create initial population of a fixed size;
do {
    choose parent1 and parent2 from population;
    offspring = Crossover(parent1, parent2);
    Mutation(offspring);
    LocalOptimize(offspring);
    Replace(population, offspring);
} until (stopping condition);
return the best individual;
```

Figure 11: The outline of the hybrid genetic algorithm

network has the same length as that of the network corresponding to v and the number of its n-pairs is smaller than $|NP(v)|$.

4. A vertex in $\mathcal{RN}(v) \cup \mathcal{EN}(v)$ that has the smallest number of n-pairs.

When a tie occurs in 4, we give favor to the comparators that are inserted in the front side.[1]

4 GA Framework

We used a typical hybrid steady-state genetic algorithm. Figure 11 shows the outline of the hybrid genetic algorithm. To evolve the parallel layers properly, we used a new encoding scheme, and devised a new crossover and mutation strategy. We describe the details of our genetic algorithm in the following.

- **Encoding:** Each sorting network is represented by a chromosome. Figure 12 shows the structure of a chromosome. A chromosome is composed of a fixed number of parallel layers and one supplemental layer. Each gene is represented by a pair of input buses and corresponds to a comparator. Each parallel layer consists of a bounded number of independent comparators and the supplemental layer consists of an unlimited number of comparators.

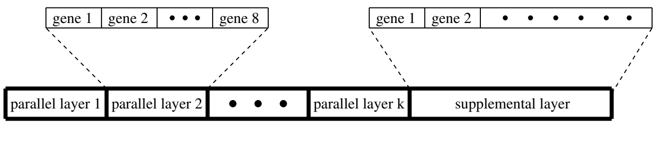


Figure 12: The structure of a chromosome

The order of the layers in the chromosome is trivially the same as that of the layers of the sorting network; and the order of the genes in each layer in the chromosome is the same as that of the comparators in the corresponding layer of the sorting network. The number of genes in each parallel layer is at most half the number of input buses. (In a 16-bus network, each parallel layer has at most 8 genes.) There is no limit to the number of genes in the supplemental layer. After all, there is no limit to the number of genes in a chromosome.

In our GA, only the genes in the parallel layers are used for crossover. On the other hand, the genes in the supplemental layer are used only for the evaluation of fitness. This is basically a type of Baldwinian hybrid GA [12] [16].

- **Initialization:** We set the population size to be 50. This is fairly smaller than Drescher's [7] and Juillé's [13]. For each parallel layer in a chromosome, we randomly generate independent comparators. The number of independent comparators are chosen between quarter and half the number of input buses. We then perform edit and repair processes to make the chromosome valid.

- **Parents Selection:** The fitness value F_i of chromosome i is calculated as follows:

$$F_i = (\frac{1}{L_i} - \frac{1}{L_w}) + (\frac{1}{L_b} - \frac{1}{L_w})/3$$

where

L_w : the length of the worst (longest) chromosome in the population,
L_b : the length of the best (shortest) chromosome in the population, and
L_i : the length of chromosome i.

Each chromosome is selected as a parent with a probability proportional to its fitness value. This is a typical proportional selection scheme.

- **Crossover:** As mentioned, we consider only the parallel layers of the two parents in crossover. We perform one-point crossover with each pair of layers independent of the other layers. Since the numbers of genes in the two layers are usually not the same, the traditional one-point crossover is not possible. Thus, we generate a cut point on the "relatively" same position in the parents. Figure 13 shows an example of the crossover. Few parallel layers of offspring made in this way are usually valid. We convert each layer to a valid one by removing comparators until there is no comparator

---

[1] Let rand[a,b] be a random integer in [a,b]. When the number of ties is t, we select the k^{th} comparator from the front in the order of inserted positions such that $k = \text{rand}[1, \text{rand}[1,t]]$.

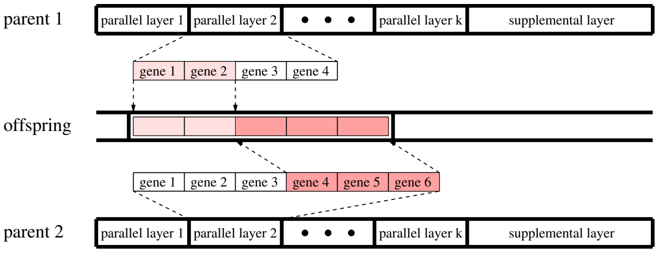


Figure 13: An example of crossover

that shares the same bus with another comparator in the layer. This crossover helps handle each parallel layer as a unit.

- **Mutation:** We randomly select each comparator with a low probability (P=0.07) in each layer and change one of the input buses at random. If there exists a comparator in the same layer that shares the changed bus, we exchange the buses of the two comparators.
- **Local Optimization:** As mentioned in Section 3, we perform the edit and repair processes to an offspring after mutation.
- **Replacement:** We replace the inferior of the two parents if the offspring is not worse than both parents. Otherwise, we replace the worst member of the population. This scheme is a compromise between preselection [5] and GENITOR-style replacement [17], and showed successful results in [4].

5 Experimental Results

Experiments were performed on an Intel Pentium III 866 MHz. As mentioned, we fixed the first 32-comparators of networks and evolved the population of networks using a genetic algorithm. In particular, we used a population size of 50, significantly smaller than other studies [11] [7] [13]. Table 2 shows the number of 60-comparator sorting networks (of quality equivalent to the best known) that we found in 15 minutes in 100 trials, according to the number of the parallel layers that we used for the GA. When we used no more than two parallel layers, we could not find any 60-comparator sorting network from 100 trials. The probability that the GA found a 60-comparator sorting network showed a bitonic distribution with respect to the number of parallel layers.

In particular, when we used 4 parallel layers, the GA found 60-comparator sorting networks in 2 to 15 minutes in all of the 100 trials. Table 3 summarizes the experimental results and the environments of Hillis [11], Drescher [7], Juillé [13], our previous study [6] and this study. When we examined the 100 networks with 60 comparators that we found using 4 parallel layers, there were 86 distinct sorting networks (of course, ignoring the sequences in the same parallel step). Table 4 classifies the 86 sorting networks according to the number of parallel steps. We present in Figure 14 one of the 60-comparator sorting networks that we found with 10 parallel steps.

Table 4: The number of distinct sorting networks according to the number of parallel steps

# of parallel steps	10	11	12	Total
# of sorting networks	26	25	35	86

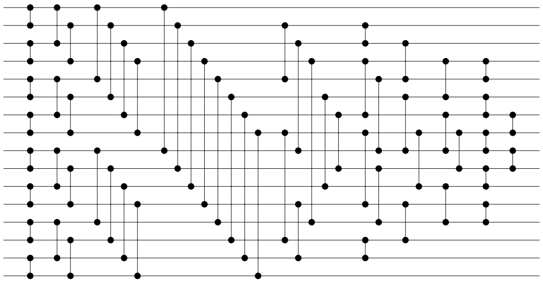


Figure 14: A 60-comparator sorting network with 10 parallel steps

6 Conclusion

We considered the sorting network problem space as a graph. This viewpoint provided us with a new insight into the problem. From this, we could see the problem space more formally and approach the problem graph-theoretically.

We presented the concept of o-pairs and n-pairs. This concept gave us a viewpoint that considers each comparator at a fairly low level. From this viewpoint we could find the redundant comparators of a network and could select the promising comparators to repair an invalid network.

Also we presented the concept of parallel layers and described the importance of the use of the parallel layers in the processes of the local optimization and the genetic algorithm. We devised a new local optimization heuristic. And, to evolve the parallel layers properly, we provided a reasonable search range for the local optimization heuristic. Following the same logic, we proposed a new encoding scheme in our genetic algorithm and devised new crossover and mutation operators accordingly.

When the local optimization heuristic and the genetic algorithm were combined, they showed strong synergy.

Table 2: The number of 60-comparator sorting networks that we found in 15 minutes in 100 trials, according to the number of the parallel layers

# of parallel layers	1	2	3	4	5	6	7	8	9	10
# of networks	0	0	74	100	98	91	66	36	21	7

Table 3: Comparison of Experimental Results and Environments

	Hillis [11]	Drescher [7]	Juillé [13]†	Choi & Moon [6]	This Study
Population size	65,536	524,288	4,096	100	50
Machine	CM-1	CM-5	Maspar MP-2 (17,000 Mips)	Pentium III 866 MHz	Pentium III 866 MHz
# of processors	65,536	64	4,096	1	1
Results	61 comparators	60 comparators, 100 % for 10 runs	60 comparators, almost 100 %	60 comparators, 76 % for 100 runs	60 comparators, 100 % for 100 runs
Execution time	5 to 50 minutes	5 to 18 minutes	5 to 10 minutes	5 to 60 minutes (average 30 minutes)	2 to 15 minutes (average 5 minutes)

†The version where the first 32 comparators are fixed

We found the best known in 16-bus networks with a fairly small time budget. To the best of our knowledge, this is the first result (with [6]) of finding 60-comparator sorting networks under a PC environment.

There are, however, remaining problems. We restricted the problem space by fixing the first 32 comparators. Although it is often believed that the global optimum will contain the fixed 32-comparators, nobody has been able to prove it. The global optimum may be in the restricted space, outside it, or in both of them. We are currently working on the theoretical justification of the restriction. This study includes experiments using the model without fixing any comparators. We would also consider problems of higher order than 16-bus in further studies.

Acknowledgements

This work was supported by Brain Korea 21 Project.

References

[1] K. E. Batcher. A new internal sorting method. *Goodyear Aerospace Report GER-11759*, 1964.

[2] R. K. Belew and T. Kammayer. Evolving aesthetic sorting networks using developmental grammars. In *Fifth International Conference on Genetic Algorithms*, page 629. Morgan Kauffmann, 1993.

[3] R. C. Bose and R. J. Nelson. A sorting problem. *Journal of ACM 9*, pages 282–296, 1962.

[4] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.

[5] D. Cavicchio. *Adaptive Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor, MI, 1970. Unpublished.

[6] S. S. Choi and B. R. Moon. A graph-based approach to the sorting network problem. In *Congress on Evolutionary Computation*, 2001.

[7] G. L. Drescher. Evolution of 16-number sorting networks revisited. Unpublished manuscript, 1994.

[8] J. R. Koza et al. Evolving sorting networks using genetic programming and the rapidly reconfigurable Xilinx 6216 field-programmable gate array. In *31st Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 404–410, 1998.

[9] R. W. Floyd and D. E. Knuth. Improved constructions for the Bose-Nelson sorting problem. *Notices of the Amer. Math. Soc. 14*, page 283, 1967.

[10] M. W. Green. Some improvements in nonadaptive sorting algorithms. Technical report, Stanford Research Institute, Menlo Park, California, 1969.

[11] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II.* Addison Wesley, 1995.

[12] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(3):495–502, 1987.

[13] H. Juillé. Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces. In *Sixth International Conference on Genetic Algorithms*, pages 351–358, 1995.

[14] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison Wesley, 1973.

[15] G. Shapiro. In D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, 1973.

[16] G. G. Simpson. The Baldwin effect. *Evolution*, 7:110–117, 1953.

[17] D. Whitley and J. Kauth. Genitor: A different genetic algorithm. In *Rocky Mountain Conf. on Artificial Intelligence*, pages 118–130, 1988.

Evolving Heterogeneous Neural Networks for Classification Problems

André L. V. Coelho
coelho@dca.fee.unicamp.br
DCA-FEEC-Unicamp
Campinas/SP – Brazil

Daniel Weingaertner
danielw@dca.fee.unicamp.br
DCA-FEEC-Unicamp
Campinas/SP – Brazil

Fernando J. Von Zuben
vonzuben@dca.fee.unicamp.br
DCA-FEEC-Unicamp
Campinas/SP – Brazil

Abstract

This paper describes research investigating the behavior of feedforward neural networks with different neuron types when applied to classification problems. For this purpose, a hierarchical evolutionary technique (HCGA) was employed towards the automatic design of heterogeneous neural nets. In an upper level, a genetic algorithm keeps in charge of building the net topology by choosing its hidden layer neurons (possibly with distinct features). Neural nets compete against each other across the GA generations. In a bottom level, a coevolutionary approach was selected in order to train the network by adjusting both the activation function parameters of the hidden neurons and its incoming input weights. This tuning is done by means of a cooperative process where the neurons receive their fitnesses according to the average fitness of the networks in which they participate.

1 INTRODUCTION

Multi-Layer Perceptron (MLP) and Radial Basis Function (RBF) networks are the most common feedforward artificial neural networks (ANNs) used on classification, recognition and prediction problems. Typically, ANNs are composed of a predefined number of similar neurons in their hidden layer. Despite the existence of some rules of thumb to indicate this parameter in accordance with the available training data, such empirical considerations are failure-prone, preventing their applicability in all cases.

In this context, designing ANNs through simulated evolution has been showing its effectiveness as an automatic alternative to manual configuration. The designer is no more required to provide expertise knowledge about the problem to be handled and its representation into the neural architecture (Whitley, 1995). Moreover, such technique is very adaptable to dynamic changes in the environment and its employment has brought improvements on a range of aspects, such as better network performance, increased robustness, faster training, and better utilization of the computational resources (parsimony). As a multitude of different net architectural solutions exists for each problem domain, finding the best configuration (or just a high quality one) can be regarded as well as a searching problem, where the aforementioned aspects could be considered as guide parameters for assessing and leveraging an evolving population of neural prototypes.

By other means, although the node activation function has been shown to be an important parameter for the ANN configuration—it is typically the source of non-linearities responsible for the approximation capabilities—, fewer works have explored such fact while devising new neural net topologies. Currently, there are no extensive results reported in the literature contemplating the actual effectiveness of employing Gaussian or logistic transfer functions in all neurons at the hidden layer. This tendency is followed, generally, for simplification purposes.

However, if the mapping function of each neuron could be automatically defined, better results regarding the net performance and the training convergence rate could be achieved. In this work, we attempt to investigate such conjecture by using GAs for the design of *heterogeneous neural networks*. The idea is to view each hidden unity as a peer with distinguished capabilities (different transfer node) to be explored while integrating with others in a same aggregate (neural net). Such abstraction aims at the *cooperative* combination of evolving populations of experts, mixing their qualifications towards the problem solution (Potter & De Jong, 2000) (Zhao *et al*., 2000).

For such endeavor, we have made use of a hierarchical approach, known as HCGA, composed of two evolutionary levels (Moriarty & Miikkulainen, 1996). In an upper level, a GA is used to build the net topology, that is, to choose neurons (possibly with distinct features) to take part into the net hidden layer. In a bottom level, a coevolutionary approach was selected in order to train the networks by adjusting both the activation function parameters of the hidden neurons and the incoming input weights.

In the remainder of the paper, we introduce remarks concerning the employment of evolutionary techniques for neural nets configuration, present our framework towards the automatic design of heterogeneous networks, show results from our experiments, identify future plans of work, and address some final considerations.

2 GENETIC ALGORITHMS AND COEVOLUTION

In the conventional genetic algorithm model, a population of strings (chromosomes) codifying the possible solutions for the problem at hand passes through a cyclic process in which new candidates are constantly created and evaluated according to some adequacy measure known as *fitness*. Ancestors are charged by computational operators, very much resembling natural evolutive phenomena (reproduction, selection and mutation), being progressively replaced by more adapted newcomers. The population fitness tends to converge on the course of the process and (sub-) optimal solutions are obtained at final stages.

Some problems with this model have been reported. First, it is very prone to the "local minima/maxima problem", as it depends on the configuration (search space distribution) of the initial population. Likewise, some fast convergence problems may occur if the population size is not properly set. Finally, it is not very suitable for the representation/generation of complex structures such as those composed by many entities (e.g. neural net architectures).

There is now such a trend to apply coevolution as a more effective technique towards complexity overcoming (Potter & De Jong, 1994). In this realm, two or more populations of the same or different species are optimized together, one influencing the other by some means. The coevolutionary process has a range of features (Potter & De Jong, 2000): (*i*) Each species represents a subcomponent of a potential solution; (*ii*) Complete solutions are obtained through the composition of members representing the different types of species (A given species could have more than one representative in the group.); (*iii*) The credit assignment for each species is a direct function of the values of fitness associated with complete solutions in which their members take part; (*iv*) There is flexibility on defining the number of available species; and (v) A separated evolutionary algorithm (GA, ES), that may have a totally different configuration from the others, does the evolution of each species.

3 EVOLUTIONARY ALGORITHMS APPLIED TO NEURAL NETWORKS

GAs have been used in conjunction with neural networks in three major undertakings (Whitley, 1995): data preprocessing, weight optimization algorithm, and a procedure to search for a suitable neural net topology. The second synergy has been hampered mainly because of the *Competing Conventions Problem*. As the weight adjustment with GAs relies heavily on recombination, there may be many equivalent symmetric solutions to the same optimization problem, which may delay the convergence process. This can be alleviated through more appropriate crossover operators, which would try to avoid individuals showing the same cyclic genetic order of their chromosomes.

Another strategy centers upon the integration between evolutionary programming (EP) and ANNs. Liu and Yao (1996) have presented an EP-based algorithm for the tuning of ANNs with different activation function nodes. The weights are adjusted through a combination of the Backpropagation (BP) algorithm with a random search algorithm. For simplicity, the authors chose to use only the logistic and the Gaussian neurons, as they represent two broad classes of activation functions with complementary features. The resulting *generalized neural net* (GNN), as they call, resembles very much what we name here as heterogeneous neural networks, albeit our approach gears towards the employment of a hierarchical coevolutionary genetic algorithm (HCGA) and is open to other types of neuronal behaviors.

Moreover, there is currently a tendency to merge constructive and evolutionary approaches. The former follow the general strategy of starting up with a minimal architectural template and then going on annexing new units and adjusting them in accordance with the others already embodied, until a desired solution could have been found (Parekh *et al.*, 1998). In such realm, Ensley and Nelson (1992) implemented another version of the Cascade-Correlation (CasCor) in which it is possible to design neural nets through the competitive fusion of neurons with different types of mapping functions. At each step, the new CasCor selects from a predefined finite set of transfer functions the one which best complements the work performed by the others.

Iyoda and Von Zuben (1999) have also attempted to analyze the impact of configuring ANNs with different activation functions at the hidden layer by proposing an evolutionary hybrid architecture inspired by another constructive method (Projection Pursuit Learning - PPL). Such approach also incorporates distinct composition functions in the output layer (additive and/or multiplicative), guiding to a higher efficiency on the combination of the mapping efforts realized by the hidden neurons. Their algorithm relies only on a classical GA, not directly promoting for the cooperation among the units.

By other means, Potter (1992) conceived the idea of replacing the gradient descent procedure typically used to adjust the unit placement by a competitive genetic algorithm. Therefore, successive runs of the GA can optimize successive hidden units. The competing entities, thus, are individual units rather than whole networks. The work depicted here follows another direction: To promote the cooperation of these building blocks by means of coevolution.

The roll of aspects indicated in the previous section for the coevolutionary approach seems very suited for simulating cooperation and/or competition behavior among neural entities. Following this premise, Whitehead and Choate (1996) have devised a cooperative-competitive genetic scheme for the automatic specification of two RBF parameters, centers and variances, when applying such kind of feedforward neural net to time series prediction. In such attempt, a GA

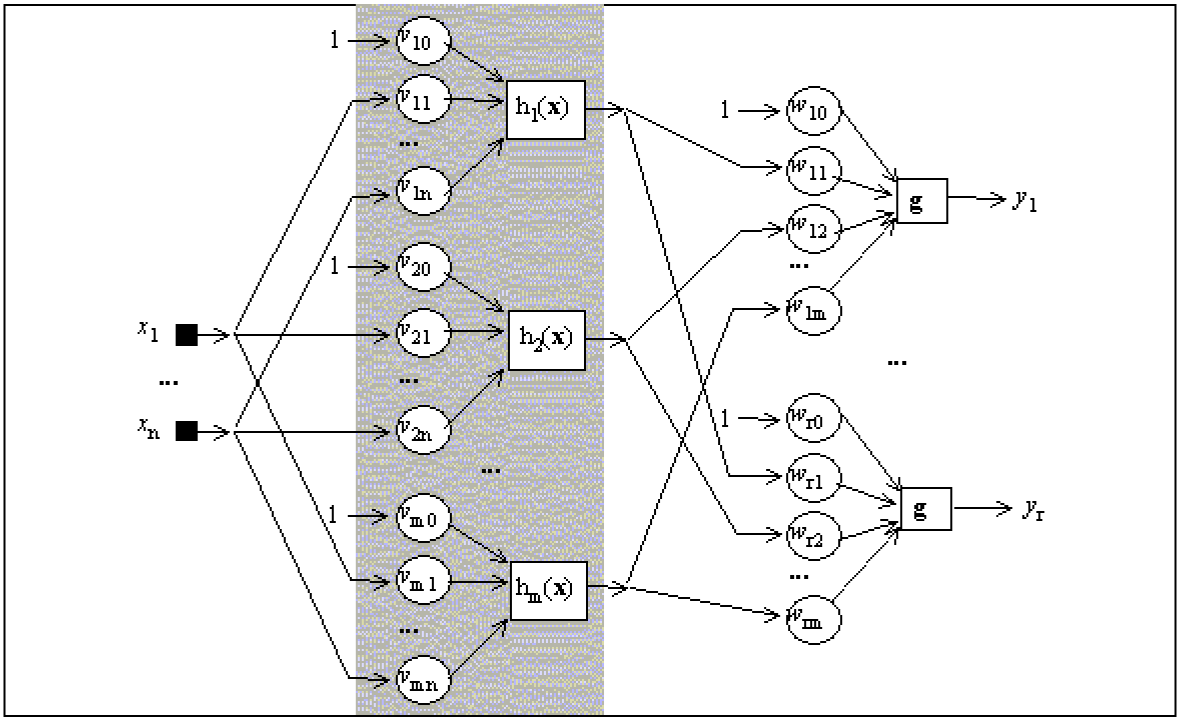


Figure 1: Architecture of the heterogeneous neural network to be evolved by the hierarchical approach. The shadowed box indicates the components to be optimized by the coevolutionary process: neurons with different activation functions and the associated weights.

operates on a population of competing basis functions $\phi_i(||\mathbf{x}-\mathbf{c}_i||/d_i)$, where $||\cdot||$ is typically the Euclidian norm and $\mathbf{x}$, $\mathbf{c}_i$, d_i are the input patterns vector, the radial basis center and a scaling factor, respectively. The whole population corresponds to a single RBF net. Hence, individual RBFs, although competing with each other, are forced to cooperatively model the desired function over the domain of interest, sharing the mapping liabilities.

Through another line of reasoning, Zhao and others (Zhao, 1997) (Zhao *et al.*, 2000) have applied cooperative coevolution to modular neural networks, devising an algorithm in which each module is individually optimized by a separate GA. In this model, the blend of neural building blocks comes to be a coarser alternative when compared with the previous one.

Finally, it is worth to mention the prominent evolutionary methodology developed by Moriarty and Miikkulainen (1996) (1998). SANE relates to a "symbiotic adaptive neuro-evolution system", in which a population of homogeneous neurons is coevolved to compose a neural net devoted to be deployed on dynamic environments. This hierarchical solution attempts to optimize the neural topology by two means. First, since neurons are recognized as functional building blocks, their ensemble can be more accurately performed. Second, since no neuron is evaluated only by its own capabilities, but rather by the qualities of the groups in which it takes part, evolutionary pressure exists to evolve several complementary neuron types. The architecture shown in the next section was inspired by such guidelines.

4 HCGA APPROACH

Here, we first present the main design deliberations taken during the conception of the HCGA approach, when applied to heterogeneous neural nets. Then we assess its adequacy and performance through a series of benchmarking results over well-known pattern classification problems.

4.1 HCGA ARCHITECTURE

Figure 1 shows the heterogeneous neural network model behind the HCGA proposal. It resembles very much a typical feedforward neural net; the main difference lies on the hidden layer as its neurons may have distinct activation functions ($h_j(\cdot)$). These functions are chosen from the delimited candidate set (which is extensible) presented in Table 1. The reason behind selecting such candidates is that they comprehend a broad parcel of those usually employed in the construction of neural nets, ranging from simple behaviors to various degrees of non-linearities. Besides the logistic (hyperbolic tangent) and Gaussian functions (as employed by Liu and Yao (1996)), we opted to make use of steps, polynomials and other Gaussian-alike shapes.

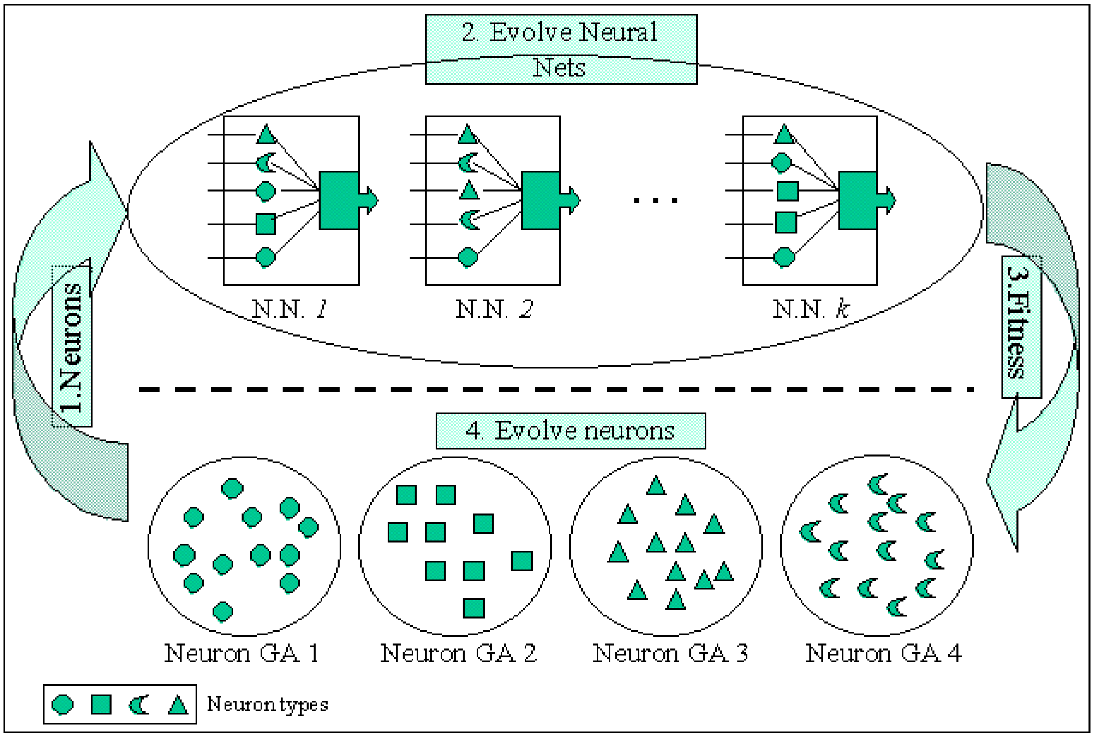


Figure 2: Process of hierarchical evolution of the neural networks.

Table 1: Set of activation function candidates.

CANDIDATE	ACTIVATION FUNCTION
Linear	$f_1 = x$
Step	$f_2 = 1$ (if x > 0) $f_2 = 0$ (otherwise)
Hyperbolic Tangent	$f_3 = \frac{e^{sx} - e^{-sx}}{e^{sx} + e^{-sx}}$
Gaussian	$f_4 = e^{\frac{-x^2}{2\sigma^2}}$
Cauchy	$f_5 = 1 + x^2 / 2\sigma^2$
Multiquadric	$f_6 = \sqrt{f_5}$
Inverse Multiquadric	$f_7 = 1 / f_6$
Polynomial	$f_8 = x^n$

As illustrated, ***x*** and ***y*** represent, respectively, the input vector of the training/testing patterns and the resulting net output. The output neurons have all the same aggregation function *g(·)*, implemented here as an additive linear combination (***Σ*)**, and the number of hidden neurons is predetermined. There are two sets of weighted connections, namely, ***V*** and ***W***. The latter is optimized during a supervised configuration process, through stimulus-response pairs, by means of the Least Mean Square (LMS) method. The adjustment of the weights in *W* is defined in order to solve the following optimization problem:

$$\min_{W} \|\boldsymbol{y} - \boldsymbol{s}\|^2,$$

where $\|\cdot\|$ is the Euclidian norm and

$$\boldsymbol{y} = \boldsymbol{HW} = \begin{bmatrix} \mathbf{h}_1(x_1) & \mathbf{h}_2(x_1) & \dots & \mathbf{h}_m(x_1) \\ \mathbf{h}_1(x_2) & \mathbf{h}_2(x_2) & \dots & \mathbf{h}_m(x_2) \\ . & . & . & . \\ \mathbf{h}_1(x_p) & \mathbf{h}_2(x_p) & \dots & \mathbf{h}_m(x_p) \end{bmatrix} \boldsymbol{W}.$$

Given the training set $\{(\mathbf{x}_i, \mathbf{s}_i)\}_{i=1}^{p}$, the LMS will then try to minimize the sum of the squared errors produced by each of the *p* input-output patterns. The *H* matrix is obtained, in this case, after the definition of *m* transfer functions chosen among the candidates given by Table 1 and applied to the *p* patterns thereupon. The optimal solution for the output weights is given by

$$\boldsymbol{H^T H W = H^T s \Rightarrow W = (H^T H)^{-1} H^T s},$$

where $(H^T H)^{-1} H^T$ is the *pseudo-inverse* of *H*, which shall only exist if *H* has a non-deficient rank.

By other means, the shadowed box in Fig. 1 indicates the components of the heterogeneous net to be optimized directly by the HCGA approach. As depicted in Fig. 2, we conceived a hierarchical evolutionary architecture composed of two levels. In a bottom level, a co-evolutionary approach was selected in order to train the networks by adjusting both the activation function parameters of the hidden neurons and the incoming input weights. In order to promote the cooperative behavior

Table 2: System configuration parameters: NNET refers to the upper level GA and NEURON to the bottom level GAs.

Parameter	Meaning	Value
NNET_POPSIZE	Number of networks in GA	50
NNET_MAX_GEN	Maximum number of generations for the neural nets	20
NNET_EXPANTION_RATE	Percentage of new elements to be inserted in the intermediary population during the neural nets refinement phase	50%
NNET_ELITIST	Number of best nets copied to next generation (elitist selection)	40%
NNET_SURVIVAL_RATE	Number of best nets whose neurons are not changed on next generation	5%
NNET_CROSSOVER_CHANCE	Chance of crossover occurrence	50%
NNET_MUTATION_CHANCE	Chance of mutation occurrence	25%
NNET_MAX_MUTATION_SIZE	Maximum number of exchanged neurons	2
NNET_NUM_HID_NEURONS	Number of neurons at the hidden layer	15
NEURON_POPSIZE	Number of neurons in each GA	20
NEURON_MAX_GEN	Number of generations for the neurons (and the whole algorithm)	50
NEURON_EXCHANGE	When to switch between selection operators (in percent of generation)	35%
NEURON_ELITIST	Number of best nets copied to next generation (elitist selection)	40%
NEURON_BICLASSIST1	Number of best nets copied to next generation (biclassist selection)	30%
NEURON_BICLASSIST2	Number of worst nets copied to next generation (biclassist selection)	20%
NEURON_CROSSOVER_CHANCE	Chance of crossover occurrence	25%
NEURON_MUTATION_CHANCE	Chance of mutation occurrence	1%
NEURON_NON_FITNESS_HIST	Maximum number of generations until the replacement of a particular neuron that has not been chosen for the composition of any net	10

among the neurons that participate in a given net (aiming at the optimized sharing and division of responsibilities) and, at the same time, detect the most promising entities across the various nets, each unity receives its fitness according to the average fitness of the networks wherein it engages. Each kind of neuron can be viewed as a distinct evolving species and is allotted to a separate GA customized to represent its attributes.

In an upper level, a conventional GA is used to build the net topology, that is, to choose neurons (possibly with distinct features) to compose the hidden layer. The same neuron, thus, is allowed both to integrate various neural architectures and to appear more than once in the same structure. Neural nets compete against each other across generations and are refined through the iterated LMS optimization process applied to their output weights.

Appendix A is dedicated to the HCGA pseudo-algorithm. The number of generations considered for benchmarking purposes relates to the upper GA cycles. The initial populations of both kinds of GA (upper and bottom) are set in a random manner (zero-average and uniform distribution). At the upper level, however, we decided to allow for the creation of new individuals in an intermediary population as a means to increase the diversity through generations. This intermediary population is also randomly generated.

At each cycle of the upper level GA, the evaluation of the neural net individuals is made over a different pattern test set, in such a way to incorporate their degree of generalization. Their fitness is inversely proportional to the mean squared error produced over the test patterns and the selective pressure is elitist generation-based, that is, the *k* better individuals from the last generation are not replaced by less adapted offspring (Hancock, 1994). To allow the net refinement phase, we decided to employ two selection operators, one for the progressive fine-tuning and the other for actual best net maintenance, whose parameter rates are given in Table 2 (NNET_ELITIST and NNET_SURVIVAL_RATE).

The chromosomes associated with neural nets are codified as follows: Each gene represents a logical link to a given neuron pertaining to a certain bottom GA. In this way, it is possible to have the same neuron appearing more than once in a given net. As the order of the neurons in the hidden layer is irrelevant, the uniform mutation operator was employed as it allows for the simultaneous swapping of *n* neurons at a time (for us, *n* was limited to 2). The recombination operator used was the *one-cut* crossover.

At the bottom level of evolution, each of the possible activation functions has an associated GA. By this means, it is possible to codify the peculiarities of each transfer function in a proper chromosome template. The fitness of the neural nets (set at the upper level) is propagated to all neurons that integrate them; in this way, the fitness of a neuron is given by the average fitness of the nets they participate. (Neurons that do not take part in any neural topology for NEURON_NON_FITNESS_HIST GA cycles are replaced.) Typically, the codification of the neuron chromosomes comprehends two slots: one for the input weights and other for the neuron's function parameters. The mutation and crossover operators actuate on one of those two slots each time, with equal probability.

4.2 RESULTS

The two pattern classification problems considered here were obtained from the PROBEN1 benchmarking

Table 3: Number of neurons of each type on the best network for each data set. **C#** stands for the *Card* data set and **H#** for the *Heart* data set.

	C1	**C2**	**C3**	**H1**	**H2**	**H3**
Linear	3	1	2	2	5	4
Step	1	6	3	3	3	7
Hyperbolic Tangent	1	2	3	2	-	1
Gaussian	3	2	4	2	2	1
Cauchy	2	-	1	3	1	-
Multiquadric	2	2	1	2	2	-
Inverse Multiquadric	3	2	1	1	1	2
Polynomial	-	-	-	-	1	-

repository (Prechelt, 1994), allowing the comparison of our proposal with others. The first problem, *Card*, refers to the task of approving or not the delivery of credit cards to particular customers, taking into account their profiles. The database has 690 samples of possible customers (patterns), with 51 input parameters (net input attributes) and two possible output responses (yes or no). 44% of the customers have good profiles and there are some absent data referring to some attributes, something that hinders the classification process. We used 518 samples to train the networks and 172 to test them. Those samples were used in 3 different orders (predefined in PROBEN1).

The *Heart* problem lies in the area of predicting cardiac diseases by observing some clinical cases (patients' health conditions). In this repository, there is a bunch of 920 patient samples (patterns) each composed of 35 inputs (personal data) and two outputs (prone to cardiac problems or not). Some attributes are also missing, making the decision even harder. We used 690 samples for training and 230 to test the networks. Here again we used the samples in 3 different, predefined orders.

Table 3 shows the configuration of the best network evolved for each problem. As can be noted, some kinds of neurons appeared more frequently than others. This is in line with what is expected: Since we are facing a classification problem, a Step neuron is typically more suitable for such a task than a Polynomial would be.

Figure 3 shows graphically the fitness of the best network and the average fitness of all networks for each data set along generations. The best network converges very fast due to the elitist selection used by the GA (Hancock, 1994). The average fitness oscillates continuously along generations because every time the neurons are evolved the networks owning them have to be readjusted.

Table 4 compares the results achieved by many neural network architectures (Ribeiro & Vasconcelos (1999)) and our HCGA approach for the same data sets. HCGA performs better than all other approaches in most of the cases, showing its great generalization capabilities when applied to classification problems. For instance, the HCGA improvement over the second best ANN for the Card-3 problem reaches nearly 30%, taking into account the classification error rate.

Liu and Xao (1996) also achieved good results when comparing their GNN approach with common MLPs and Gaussian MLPs in classification problems. Through a series of experiments, the evolved GNN performed better, exhibiting lower means of the average test set error against the ones produced by the MLP and Gaussian MLP. Nevertheless, the benchmarking repository used for their analysis was a modified version of the *Heart* database, composed only of 303 data samples, with only 270 patterns effectively exploited.

Iyoda and Von Zuben (1999), as well, attained good outcomes when comparing the performance of their hybrid architecture against the typical MLP and PPL approaches in approximation problems. Nonetheless, the computational costs related to their proposal constantly reached huge amounts, something that may become a bottleneck when applying the technique to hard problems.

Table 4: Compared results (percentage of correctly classified patterns) between HCGA and other ANNs.

	C1	C2	C3	H1	H2	H3
MLP	86.00	81.00	81.00	80.00	82.00	76.00
RBF	88.00	82.00	83.00	82.00	82.00	79.00
CasCor	84.00	79.00	82.00	82.00	80.00	74.00
Tower	84.88	78.49	79.65	78.70	78.26	70.87
Pyramid	86.05	79.07	79.07	79.57	78.26	76.09
DistAl	84.35	84.06	82.90	79.13	80.87	82.61
PerCasc	87.58	86.80	84.07	82.72	82.56	80.96
Upstart	90.70	86.05	83.72	84.05	81.45	80.78
Tiling	80.23	79.05	79.07	76.52	71.64	77.39
HCGA	**90.12**	**87.79**	**88.95**	**88.26**	**85.65**	**81.30**

4.3 FUTURE WORK

Some improvements could be undertaken as future work. To combine other constructive techniques with ours in different manners seems suitable. For instance, to employ a CasCor implementation to produce a net that would serve as an initial start-up template to generate the initial population of neural nets at the upper HCGA level. This would comprise a typical sequential arrangement of strategies.

Likewise, it would also be possible to incorporate the idea established by Iyoda and Von Zuben (1999) of evolving distinct composition functions in the output layer (additive and/or multiplicative). Other enterprises include: (*i*) The possibility of also augmenting or pruning (Reed, 1994) the number of hidden neurons at each generation, given space to the arising of "extend-shrink" behaviors; (*ii*) The employment of other EAs to evolve the neuronal subpopulations; (*iii*) The expansion in the number of levels in the hierarchy; and (*iv*) The application of cooperative coevolution to more coarser building blocks.

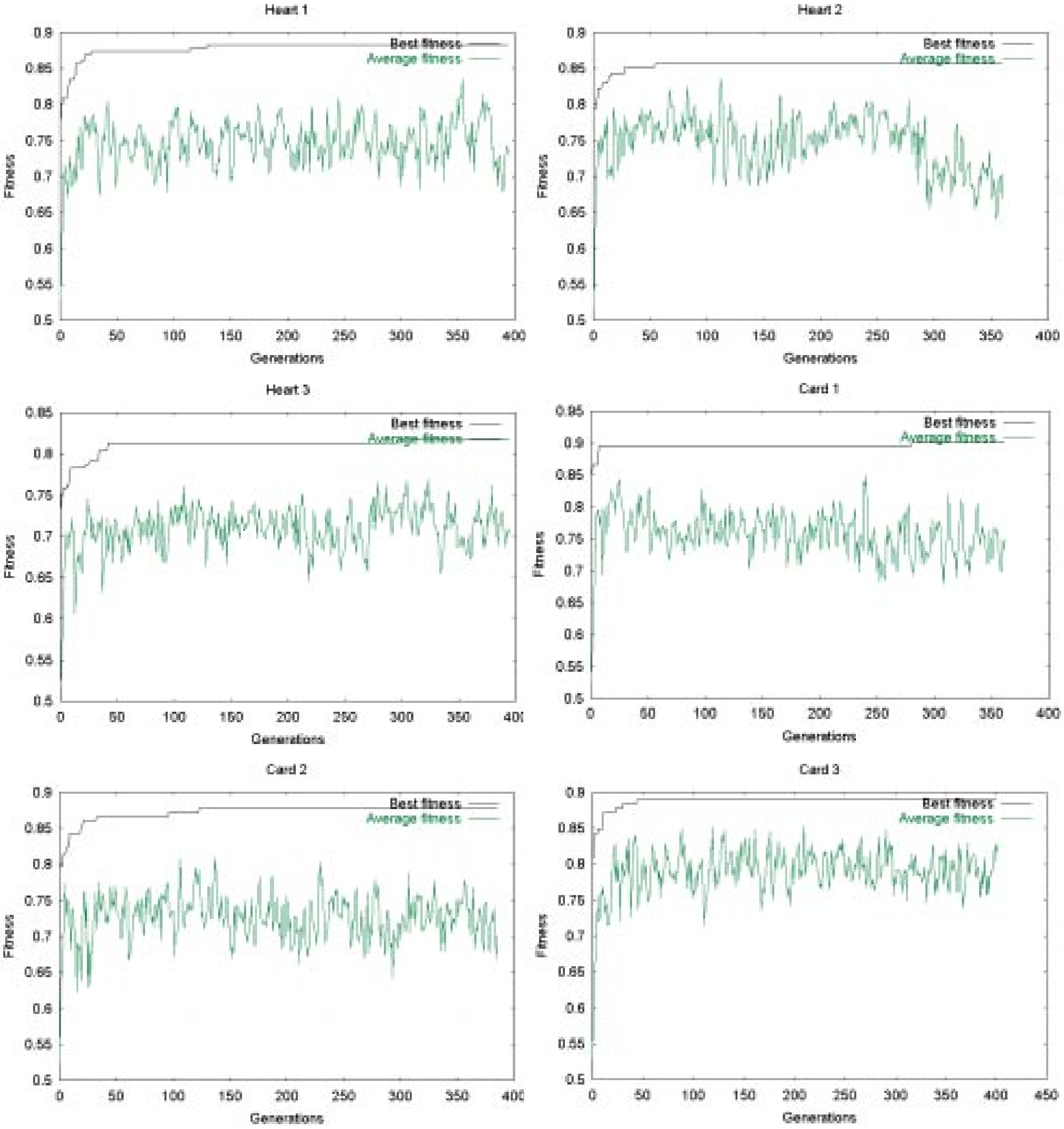


Figure 3: Results showing the evolution of the best neural network and the average fitness of the networks along generations.

5 CONCLUSION

This work has examined the suitability of merging together into the same framework various promising approaches proposed recently for the configuration of neural net architectures, particularly involving coevolution. The resulting model (HCGA) comprehends a new hierarchical-based evolutionary scheme devoted to the progressive assembling of heterogeneous neural structures. This blending strategy was assessed through a series of benchmarking tests over classification problems. The findings obtained so far corroborate other results already presented in the literature, showing that HCGA constitutes a promising design strategy in the direction of fully automatic adjustment of an extended set of neural networks parameters.

Acknowledgments

CNPq and **CAPES** have partially sponsored this research through Ph.D. and MSc scholarships to the first and second authors, respectively.

References

D. Ensley and D. Nelson (1992). Extrapolation of MacKey-Glass Data using Cascade Correlation. *Simulation* **58**(5), 333-339.

P. Hancock (1994). An Empirical Comparison of Selection Methods in Evolutionary Algorithms. *Proceedings of the Third Parallel Problem Solving from Nature Conference*, *Lecture Notes in Computer Science 865*, 80-94, Springer.

E. Iyoda and F. Von Zuben (1999). Evolutionary Hybrid Composition of Activation Functions in Feedforward Neural Networks. *Proceedings of INNS-IEEE International Joint Conference on Neural Networks*, Vol. 6, 4048-4053.

Y. Liu and X. Yao (1996). Evolutionary Design of Artificial Neural Networks with Different Nodes. *Proceedings. of the 1996 IEEE International Conference on Evolutionary Computation*, 670-675, Japan, May.

D. Moriarty and R. Miikkulainen (1996). Efficient Reinforcement Learning through Symbiotic Evolution. *Machine Learning* **22**, 11-33. Kluwer Academic Publishers.

D. Moriarty and R. Miikkulainen (1998). Hierarchical Evolution of Neural Networks. *Proceedings. of the 1998 IEEE International Conference on Evolutionary Computation*, 428-433.

R. Parekh, J. Yang, and V. Honavar (1998). Constructive Neural Network Learning Algorithms for Multi-Category Real-Valued Pattern Classification. *TR 97-06*, Artificial Intelligence Research Group, Department of Computer Science, Iowa State University.

M. Potter (1992). A Genetic Cascade-Correlation Learning Algorithm. In L. D. Whitley and J. Schaffer (eds.), *Combinations of Genetic Algorithms and Neural Networks*, 123-133, IEEE Press.

M. Potter and K. De Jong (1994). A Cooperative Coevolutionary Approach to Function Optimization. *Proceedings of the Third Parallel Problem Solving from Nature Conference*, *Lecture Notes in Computer Science 865*, 249-257, Springer.

M. Potter and K. De Jong (2000). Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. Evolutionary Computation **8**(1):1-29.

L. Prechelt (1994). PROBEN1 – A Set of Neural Benchmarking Rules. *TR 21/94*, Universtät Karlsruhe.

R. Reed (1993). Pruning Algorithms – A Survey, *IEEE Transactions on Neural Networks* **4**(5), 740-747.

J. Ribeiro and G. Vasconcelos (1999). Constructive Neural Networks for Pattern Classification. *Annals of XIX National Congress of the Brazilian Computer Society*, Vol. 4, 53-64, Rio de Janeiro (*in Portuguese*).

B. Whitehead and T. Choate (1996). Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction. *IEEE Transactions on Neural Networks* **7**(4), 869-880, July.

D. Whitley (1995). Genetic Algorithms and Neural Networks. In J. Periaux and G. Winter (eds.), *Genetic Algorithms in Engineering and Computer Science*, John Wiley & Sons.

Q. Zhao (1997). A Co-evolutionary Algorithm for Neural Network Learning. *Proceedings. of the International Conference on Neural Networks*, vol. 1, 432-437.

Q. F. Zhao, O. Hammami, K. Kuroda and K. Saito (2000). Cooperative Co-evolutionary Algorithm – How to Evaluate a Module? Proceedings of the IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, 150-157, San Antonio, May.

Appendix A (HGCA Pseudo-algorithm)

```
1. Create initial population;
       Create neuron populations;
       Create neural net population;

2. Evolve neural nets (refinement);
    do
        Create intermediary population (expansion);
        Evaluate neural networks
            Train (adjust weights via LMS)
            Test (compute NN_error);
        Compute fitness;
            fitness ⇐– NN_error;
        Elitist Selection;
        Mutation;
        Adjust the expansion rate;

    while( subGenCount < NNET_MAX_GEN &&
             (bestFit > lastBestFit || avgFit >lastAvgFit) );
    Set survival flag on neurons of the NUM_SURVIVE
                                          best networks;
    Cutoff the exceeding neural nets;

    Propagate fitness to neurons;
        Neuron fitness ⇐ avg of NNets it participated;
 3. Evolve neurons;

    if( subGenCount < NEURON_EXCHANGE )
        Biclassist Selection;
    Else
        Elitist Selection;
        Mutation;
4. Repeat steps 2. and 3. for NEURON_MAX_GEN times
```

Multiobjective Optimization using a Micro-Genetic Algorithm

Carlos A. Coello
CINVESTAV-IPN
Depto. de Ing. Elec./Secc. Computación
Av. Instituto Politécnico Nacional No. 2508
Sn. Pedro Zacatenco, México, D. F. 07300
ccoello@cs.cinvestav.mx

Gregorio Toscano Pulido
Maestría en Inteligencia Artificial
LANIA-Universidad Veracruzana
Sebastián Camacho No. 5
Xalapa, Veracruz, México 91090
gtoscano@mia.uv.mx

Abstract

In this paper, we propose a micro genetic algorithm with three forms of elitism for multiobjective optimization. We show how this relatively simple algorithm coupled with an external file and a diversity approach based on geographical distribution can generate efficiently the Pareto fronts of several difficult test functions (both constrained and unconstrained). A metric based on the average distance to the Pareto optimal set is used to compare our results against two evolutionary multiobjective optimization techniques recently proposed in the literature.

1 INTRODUCTION

Despite the considerable volume of research on evolutionary multiobjective optimization (see for example [4, 1, 16]), until recently, little emphasis had been placed on developing efficient techniques. The usual approach has been to use a ranking procedure to classify a population of individuals based on their Pareto dominance. This ranking procedure normally consumes most of the running time of an evolutionary multiobjective optimization technique[1]. Pareto ranking is $O(kM^2)$, where k is the number of objective functions and M is the size of the population. Additionally, an extra mechanism is required to preserve diversity (some form of fitness sharing [3] is normally adopted). This generally implies the use of another process that is $O(M^2)$.

Some authors have recently addressed efficiency issues in the context of evolutionary multiobjective optimization (e.g., [10, 2]). Knowing the sources of inefficiency of traditional evolutionary multiobjective optimization techniques, several researchers have focused their recent efforts on reducing the checkings for nondominance and in the development of efficient approaches to keep diversity. Regarding the first issue, the main emphasis has been on using an external file that stores nondominated vectors found during the evolutionary process. These vectors are put back into the population at later generations (this can be seen as a form of elitism in the context of multiobjective optimization [6, 18]). Regarding the second issue, the main emphasis has been on using clustering techniques [2] or approaches based on geographical positioning of individuals in an adaptive grid [10].

Also, some researchers have suggested the use of a distributed GA in which Pareto dominance is applied only to neighbors within a certain region [13]. Such sort of approach can handle the two problems previously mentioned simultaneously. The approach is efficient because Pareto dominance is applied in parallel to small groups of individuals. Diversity does not require an extra mechanism, since it naturally emerges from the distributed population. However, to take advantage of these features of the algorithm, a parallel architecture is necessary.

Our approach was to use a GA with a very small population size and a reinitialization process (a micro-GA) to solve multiobjective optimization problems of different degrees of complexity. To validate the performance of our approach, we used a metric previously defined in the literature to compare our results against two techniques that are representative of the state-of-the-art in evolutionary multiobjective optimization algorithms: the Nondominated Sorting Genetic Algorithm II (NSGA II) [2] and the Pareto Archived Evolution Strategy (PAES) [10].

---

[1] This is obviously considering the academic test functions that most researchers have used so far. In real-world problems, most of the computational time is normally spent evaluating the fitness functions of the problem.

2 PREVIOUS WORK

The term micro-genetic algorithm (micro-GA) refers to a small-population genetic algorithm with reinitialization. The approach was derived from some theoretical results obtained by Goldberg [5], according to which a population size of three was sufficient to converge, regardless of the chromosomic length. The process suggested by Goldberg was to start with a small randomly generated population, then apply to it the genetic operators until reaching *nominal convergence* (e.g., when all the individuals have their genotypes either identical or very similar), and then to generate a new population by transferring the best individuals of the converged population to the new one. The remaining individuals would be randomly generated.

The first to report an implementation of a micro-GA was Krishnakumar [11], who used a population size of five, a crossover rate of one and a mutation rate of zero. His approach also adopted an elitist strategy that copied the best string found in the current population to the next generation. Selection was performed by holding four competitions between strings that were adjacent in the population array, and declaring to the individual with the highest fitness as the winner. Krishnakumar [11] compared his micro-GA against a simple GA (with a population size of 50, a crossover rate of 0.6 and a mutation rate of 0.001). He reported faster and better results with his micro-GA on two stationary functions and a real-world engineering control problem (a wind-shear controller task). After him, several other researchers have developed applications of micro-GAs (e.g., [8, 17]). However, the work reported in this paper represents, to the best of our knowledge, the first attempt to use a micro-GA for multiobjective optimization.

Regarding similar work, we are only aware of an approach developed by Jaszkiewicz [7] in which a small population initialized from a large external memory is used for a short period of time. However, to the best of our knowledge, this approach has been used only for multiobjective combinatorial optimization. Some could also argue that the multi-membered versions of PAES can be seen as a form of micro-GA. However, the authors of PAES concluded that the addition of a population did not, in general, improve the performance of their approach, and increased the computational overhead in an important way [10].

3 DESCRIPTION OF OUR APPROACH

The way in which our technique works is illustrated in Figure 1. First, a random population is generated. This random population feeds the population memory, which is divided in two parts: a replaceable and a non-replaceable portion. The non-replaceable portion of the population memory will never change during the entire run and is meant to provide the required diversity for the algorithm. In contrast, the replaceable portion will experience changes after each cycle of the micro-GA.

The population of the micro-GA at the beginning of each of its cycles is taken (with a certain probability) from both portions of the population memory so that we can have a mixture of randomly generated individuals (non-replaceable portion) and evolved individuals (replaceable portion).

During each cycle, the micro-GA undergoes conventional genetic operators (binary representation is used in our implementation): tournament selection, two-point crossover, uniform mutation, and elitism. After the micro-GA finishes one cycle, we choose two nondominated vectors[2] from the final population and compare them with the contents of the external memory (this memory is initially empty). If either of them (or both) remains as nondominated after comparing it against the vectors in this external memory, then they are included there (i.e., in the external memory). This is our historical archive of nondominated vectors. All dominated vectors contained in the external memory are eliminated.

The same two vectors previously mentioned are also compared against two elements from the replaceable portion of the population memory. If either of these vectors dominates to its match in the population memory, then it replaces it. Otherwise, the vector is discarded. Over time, the replaceable part of the population memory will tend to have more nondominated vectors, some of which will be used in some of the initial populations of the micro-GA.

Our approach uses three types of elitism. The first is based on the notion that if we store the nondominated vectors produced from each cycle of the micro-GA, we will not lose any valuable information obtained from the evolutionary process. The second is based on the idea that if we replace the population memory by the nominal solutions (i.e., the best solutions found

---

[2] This is assuming that we have two or more nondominated vectors. If there is only one, then this vector is the only one selected.

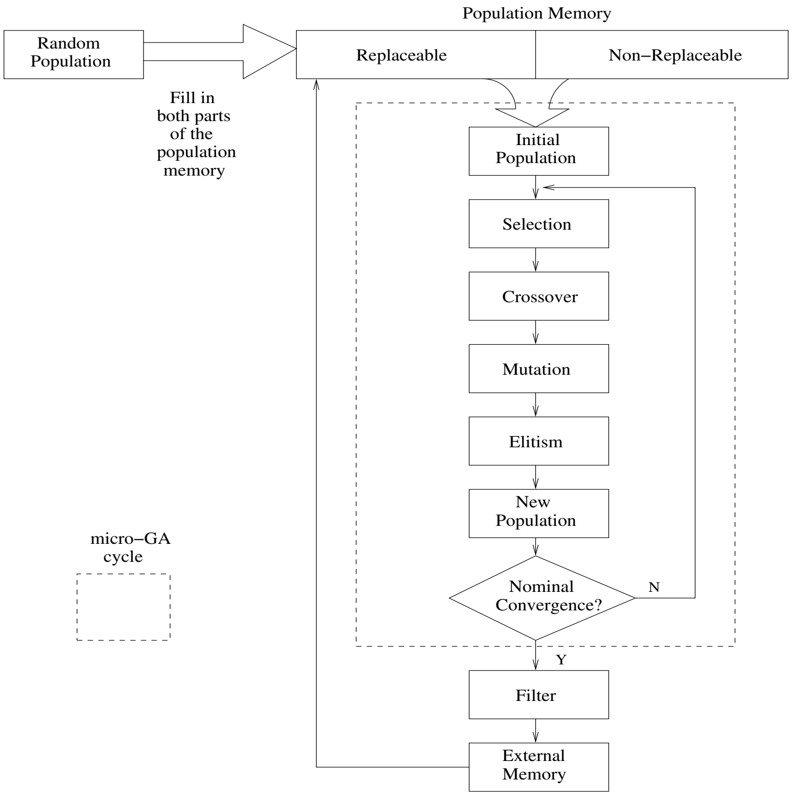


Figure 1: Diagram that illustrates the way in which our micro-GA works.

when nominal convergence is reached), we will gradually converge, since crossover and mutation will have a higher probability of reaching the true Pareto front of the problem over time. Nominal convergence, in our case, is defined in terms of a certain (low) number of generations (two to five in our case). The third type of elitism is applied at certain intervals (defined by a parameter called "replacement cycle"). We take a certain number of points from all the regions of the Pareto front generated so far and we use them to fill the replaceable memory. Depending on the size of the replaceable memory, we choose as many points from the Pareto front as necessary to guarantee a uniform distribution. This process allows us to use the best solutions generated so far as the starting point for the micro-GA, so that we can improve them (either by getting closer to the true Pareto front or by getting a better distribution).

To keep diversity in the Pareto front, we use an approach similar to the adaptive grid proposed by Knowles & Corne [10]. Once the archive that stores nondominated solutions has reached its limit, we divide the objective search space that this archive covers, assigning a set of coordinates to each solution. Then, each newly generated nondominated solution will be accepted only if the geographical location to where the individual belongs has fewer individuals than the most crowded location. Alternatively, the new nondominated solution could also be accepted if the individual belongs to a location outside the previously specified boundaries.

The adaptive grid requires two parameters: the expected size of the Pareto front and the number of positions in which we will divide the solution space for each objective. The first parameter is defined by the size of the external memory. We have found that our approach is not very sensitive to the second parameter (e.g., in our experiments a value of 15 or 25 provided very similar results). The process of determining the location of a certain individual has a low computational cost (it is based on the values of its objectives as indicated before). However, when the individual is out of range, we have to relocate all the positions. Nevertheless, this last situation does not occur too often, and we allocate a certain amount of extra room in the first and last locations of the grid to minimize its occurrence.

4 COMPARISON OF RESULTS

Several test functions were taken from the specialized literature to compare our approach. In all cases, we generated the true Pareto fronts of the problems using exhaustive enumeration (with a certain granularity) so that we could make a graphical comparison of the quality of the solutions produced by our micro-GA. Additionally, we decided to use one of the metrics defined in objective space by Zitzler et al. [18]:

$$M_1^* = \frac{1}{\mid Y' \mid} \sum_{\mathbf{d}' \in \mathbf{Y}'} \min \left\{ \parallel \mathbf{d}' - \bar{\mathbf{d}} \parallel^* ; \bar{\mathbf{d}} \in \bar{\mathbf{Y}} \right\} \qquad (1)$$

where: $Y', \bar{Y} \subseteq Y$ are the sets of objective vectors that correspond to a set of pairwise nondominating decision vectors $X', \bar{X} \subseteq X$, respectively, and X corresponds to the decision variables of the problem. It should be obvious that M_1^* gives the average distance to the Pareto optimal set. Therefore, we should aim to minimize this value (see [18] for further details).

Since the main aim of this approach has been to increase efficiency, we additionally decided to compare running times of our micro-GA against two very fast algorithms: the NSGA II [2] and PAES[3] [10].

In the following examples, the NSGA II was run using a population size of 100, a crossover rate of 0.8, tournament selection, and a mutation rate of 1/vars, where vars = number of decision variables of the problem. In the following examples, PAES was run using a depth of five, a size of the archive of 100, and a mutation rate of $1/L$, where L refers to the length of the chromosomic string that encodes the decision variables.

For constrained functions, we used a very simple approach. Whenever two individuals were compared, we checked their constraints. If both were feasible, nondominance was directly applied. If one was feasible and the other was infeasible, the feasible would dominate. If both were infeasible, then the one with the lowest amount of constraint violation would dominate the other. This same approach was used in PAES. The NSGA II has its own constraint-handling mechanism, so we did not have to implement one for it.

To allow a fair comparison of running times, all the experiments were performed on a PC with a Pentium III processor running at 650 MHz, 128 Mb of RAM and a hard drive of 15 Gbytes. Our implementation was compiled using GNU C running under Linux Red Hat release 6.2.

Several test functions were used to validate our approach, but due to space limitations, only the results corresponding to the four test functions shown in Table 1 were included in this paper. In all our experiments, our micro-GA used a crossover rate of 0.7, an external memory of 100 individuals, a number of iterations to achieve nominal convergence of two, a population memory of 50 individuals, a percentage of non-replaceable memory of 0.3, a population size (for the micro-GA itself) of four individuals, and 25 subdivisions of the adaptive grid. The other parameters used are shown in Table 2. Note that the mutation rate was always $1/L$ (L = length of the chromosomic string).

Figures 2, 3, 4, and 5, show the results produced by the NSGA II, PAES and our micro-GA in the four test functions adopted. The true Pareto fronts of each problem are also shown in each figure.

Results are summarized in Table 3. In all the unconstrained test functions used, the micro-GA obtained the lowest CPU time and the lowest value of the metric M_1^*. For the constrained test functions (such as functions three and four), the NSGA II obtained the lowest value of the metric, and the micro-GA placed second. However, note that for the third test function, the micro-GA only took a third of the running time than the NSGA II and it covered most of the Pareto front of the problem unlike the other two algorithms.

5 ANALYSIS OF RESULTS

There are a few things that we can say about the observed behavior of the three algorithms compared. The NSGA II is a very good algorithm that provides elegant solutions and a good performance (in terms of CPU time). However, we have found that in some test functions the NSGA II is not able to cover properly the whole Pareto front. We believe that its exploratory capabilities could be improved, and that its main strength is its extraordinary capability to exploit a promising region of the search space, once it finds it. This last point is in fact the main weakness of our micro-GA in its current form. However, our micro-GA has compared relatively well in terms of the metric adopted and it obtained the lowest computational costs in all the test functions used. In fact, for the case of the unconstrained test functions used, the micro-GA exhibited the best overall performance. That is also an

---

[3] Readers interested in reproducing these experiments may download the source code of the NSGA II and PAES (original versions from their corresponding authors) from the EMOO repository located at `http://www.lania.mx/~ccoello/EMOO/EMOOsoftware.html`. The code of the micro-GA is available from the authors upon request.

Table 1: Test Functions used to validate our micro-GA

TEST FUNCTION	OBJECTIVES	SOURCE
1	Min $f_1(x) = \begin{cases} -x & \text{if } x \leq 1 \\ -2+x & \text{if } 1 < x \leq 3 \\ 4-x & \text{if } 3 < x \leq 4 \\ -4+x & \text{if } x > 4 \end{cases}$ Min $f_2(x) = (x-5)^2$ $-5 \leq x \leq 10$	[14]
2	Min $f_1(\vec{x}) = \sum_{i=1}^{n-1} \left(-10 \exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2}\right)\right)$ Min $f_2(\vec{x}) = \sum_{i=1}^{n} \left(\lvert x_i \rvert^{0.8} + 5\sin(x_i)^3\right)$ $-5 \leq x_1, x_2, x_3 \leq 5$	[12]
3	Max $f_1(x,y) = -x^2 + y$ Max $f_2(x,y) = \frac{1}{2}x + y + 1$ $\frac{1}{6}x + y - \frac{13}{2} \leq 0$ $\frac{1}{2}x + y - \frac{15}{2} \leq 0$ $5x + y - 30 \leq 0$	[9]
4	Max $f_1(x,y) = (x-2)^2 + (y-1)^2 + 2$ Max $f_2(x,y) = 9x - (y-1)^2$ $x^2 + y^2 - 225 \leq 0$ $x - 3y + 10 \leq 0$	[15]

Table 2: Some of the parameters used by our micro-GA for each of four test functions (TF) used to validate our approach (the other parameters were kept constant in all our experiments)

PARAMETER	TF1	TF2	TF3	TF4
number of iterations	150	3000	2500	1500
mutation rate	0.056	0.019	0.0217	0.0192
replacement cycle (iterations)	25	50	50	100

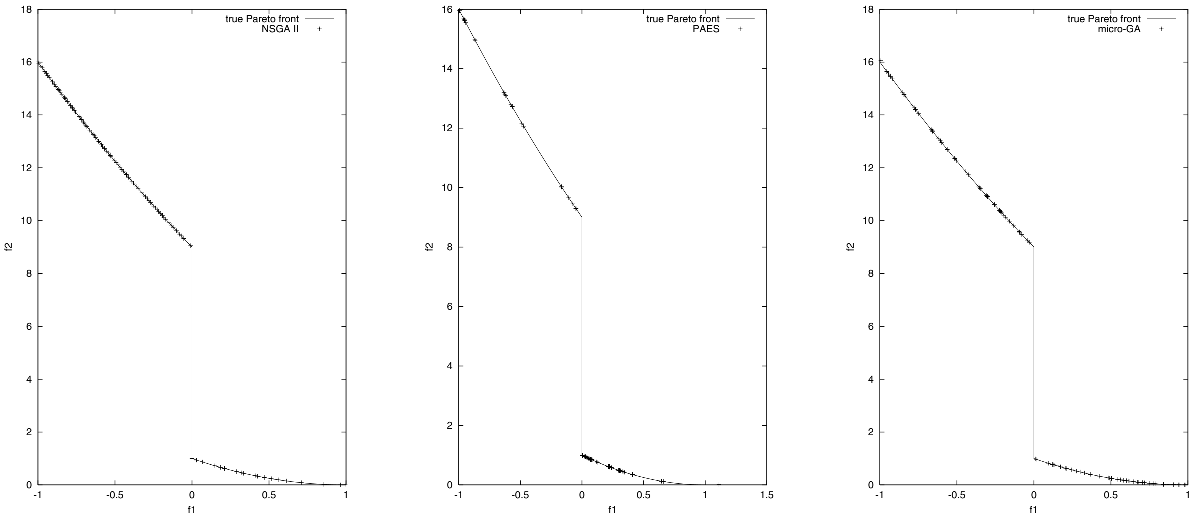


Figure 2: Pareto fronts produced by the NSGA II (left), PAES (middle), and our micro-GA (right) for the first test function

Table 3: Comparison of results. Results are reported over 20 runs

	TEST FUNCTION 1		
Perf. measure	**NSGA II**	**PAES**	**micro-GA**
average M_1^*	0.00161422	0.0675201338	0.001530162
variance of M_1^*	0.0000000200	0.0169825683	0.0000005886
average running time	0.282s	0.107s	0.017s
fitness function evals.	1,200	1,200	1,200
	TEST FUNCTION 2		
average M_1^*	0.13777005	0.42445655	0.13460185
variance of M_1^*	0.0000288497	0.0645582672	0.0000479001
average running time	6.481s	2.195s	0.704s
fitness function evals.	24,000	24,000	24,000
	TEST FUNCTION 3		
average M_1^*	0.04684924	0.399809545	0.26210439
variance of M_1^*	0.0064993289	0.2463272535	0.0622687130
average running time	6.4857s	68.937s	2.6896s
fitness function evals.	20,000	20,000	20,000
	TEST FUNCTION 4		
average M_1^*	0.2951232	5.6864776	0.4046362
variance of M_1^*	0.0015191459	384.12725166	0.0214295578
average running time	4.038s	56.6706s	3.4679s
fitness function evals.	12,000	12,000	12,000

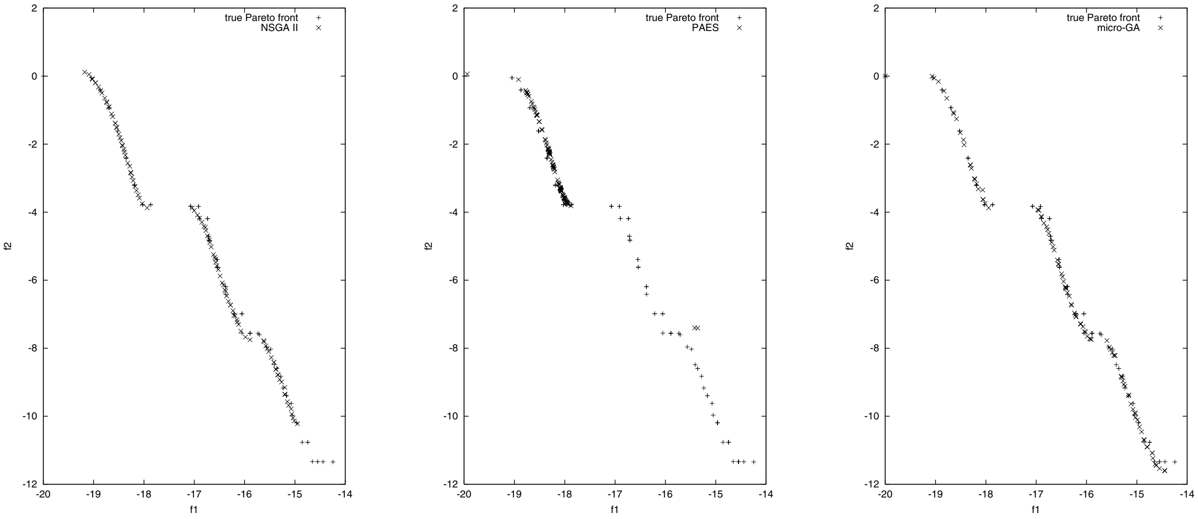


Figure 3: Pareto fronts produced by the NSGA II (left), PAES (middle), and our micro-GA (right) for the second test function

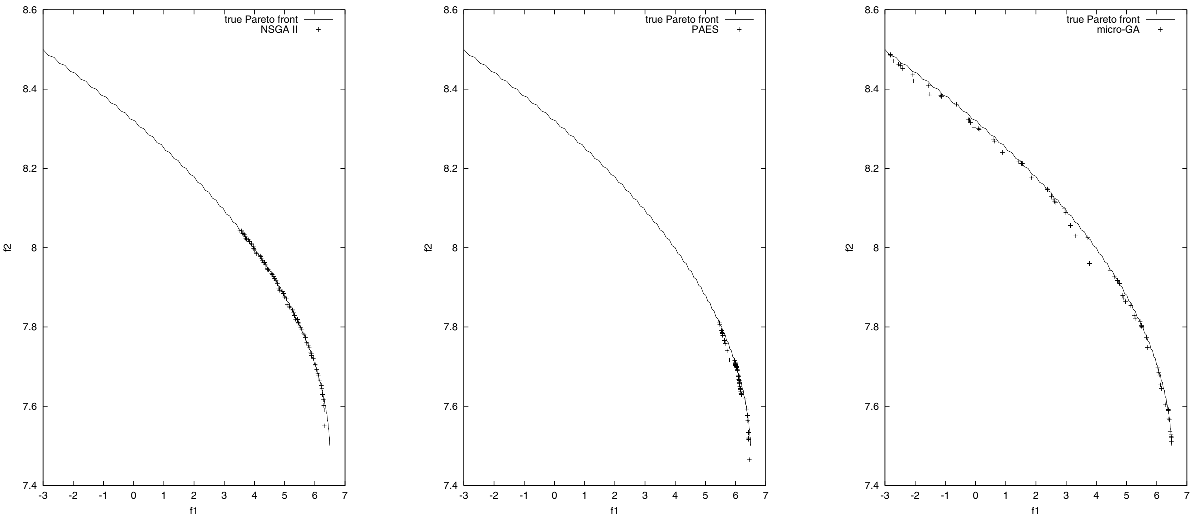


Figure 4: Pareto fronts produced by the NSGA II (left), PAES (middle), and our micro-GA (right) for the third test function

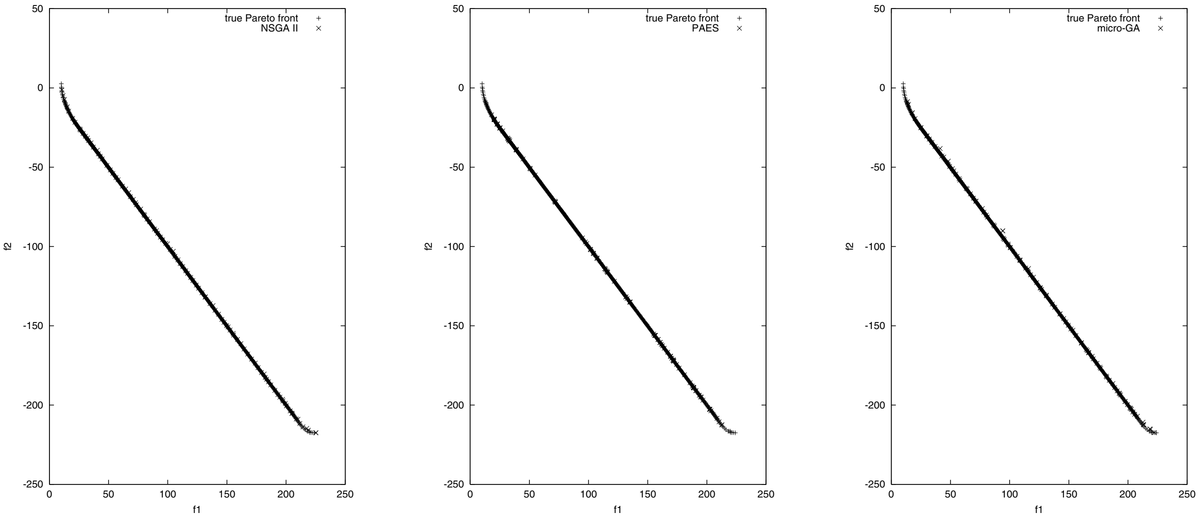


Figure 5: Pareto fronts produced by the NSGA II (left), PAES (middle), and our micro-GA (right) for the fourth test function

indicative that our approach to incorporate constraints may not be the most appropriate for the micro-GA and we are studying other techniques. Finally, PAES has difficulty with disconnected Pareto fronts, since in those cases it exhibited its worst behavior. Also, the approach used to handle constraints with PAES (the same adopted for the micro-GA) may not be the most appropriate and it probably had some impact on its performance. Nevertheless, a more comprehensive study is still necessary (using more test functions and other metrics) to derive more general conclusions.

6 THE PARAMETERS OF OUR APPROACH

Since our micro-GA uses several parameters that are not typical of evolutionary multiobjective optimization approaches, we performed several experiments to try to determine a set of values that can be used by default (i.e., when nothing about the problem is known).

The size of the external memory is a parameter that should be easy to setup, since it corresponds to the number of nondominated vectors that the user wishes to find.

Regarding the size of the population memory, we recommend to set it to 50% of the size of the external memory. The reason is that if a larger percentage is used, the number of individuals to undergo evolution becomes too large. On the other hand, if the percentage is lower, we can easily lose diversity.

For the number of iterations of the micro-GA, we found that a value between two and five seems to work well. It is important to be aware of the fact that a larger value for this parameter implies a greater CPU cost for the algorithm. However, a larger value provides Pareto fronts with a better spread. Therefore, the setup of this parameter is really a trade-off between efficiency and quality of the solutions found.

Regarding the number of subdivisions of the adaptive grid, the recommended range is a value between 5 and 100. As a default value, we suggest 25, which is the value that provided the best overall performance in our experiments. Larger values for this parameter will provide a better spread of the Pareto front, but will sacrifice efficiency and memory requirements.

For the percentage of non-replaceable memory, we suggest to use 0.3, since this value ensures that for each pair of individuals evolved, one will be randomly selected (i.e., this promotes diversity).

Finally, for the replacement cycle, we suggest to use a value between 25 and n (where n is the total number of iterations). We have used values between 25 and 200 for this parameter. However, this is a parameter that requires special attention and we intend to study its behavior in more detail to try to derive more general values within a narrower range. This value is also critical for our algorithm, because if it is too small, the algorithm may converge to a local Pareto front. If it is too large, the replacement of the population at each cycle may not be enough to guarantee the necessary diversity. So far, the value proposed has been empirically set up for each particular problem.

7 CONCLUSIONS AND FUTURE WORK

We have proposed the use of a GA with a very small population size (only four individuals) and a reinitialization process to solve multiobjective optimization problems. Our approach has been compared against the NSGA II and PAES in several test functions. In the unconstrained test functions used, our approach has been able to converge faster (in terms of CPU time) to the true Pareto front than the two other algorithms analyzed. Also, it has performed better than them in terms of a metric previously proposed in the specialized literature. In the constrained functions, however, its performance has not been as good as that of the NSGA II, although it has been better than PAES. Also, in some cases, it produced a better distribution along the Pareto front than any of the other two algorithms analyzed.

Our initial future work will be to analyze other approaches to handle constraints in our micro-GA and to study the capabilities of our algorithm to exploit a certain promising region of the search space (the main advantage of the NSGA II over our approach). We will also perform a careful sensitivity study of the parameters of the algorithm so that we can provide more general guidelines to set them up, and we also aim to eliminate some of the parameters currently used.

Acknowledgements

The authors would like to thank the four anonymous reviewers for their valuable comments that helped them improve this paper.

The first author gratefully acknowledges support from CONACyT through project 34201-A to perform this work. The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Maestría en Inteligencia Artificial of LANIA and the Universidad Veracruzana.

References

[1] Carlos A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, August 1999.

[2] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858. Springer, 2000.

[3] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[4] Carlos M. Fonseca and Peter J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[5] David E. Goldberg. Sizing Populations for Serial and Parallel Genetic Algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79, San Mateo, California, 1989. Morgan Kaufmann Publishers.

[6] Hisao Ishibuchi and Tadahiko Murata. Multi-Objective Genetic Local Search Algorithm. In Toshio Fukuda and Takeshi Furuhashi, editors, *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 119–124, Nagoya, Japan, 1996. IEEE.

[7] Andrzej Jaszkiewicz. Genetic local search for multiple objective combinatorial optimization. Technical Report RA-014/98, Institute of Computing Science, Poznan University of Technology, 1998.

[8] E.G. Johnson and M.A.G. Abushagur. Micro-Genetic Algorithm Optimization Methods Applied to Dielectric Gratings. *Journal of the Optical Society of America*, 12(5):1152–1160, 1995.

[9] Hajime Kita, Yasuyuki Yabumoto, Naoki Mori, and Yoshikazu Nishikawa. Multi-Objective Optimization by Means of the Thermodynamical Genetic Algorithm. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN IV*, Lecture Notes in Computer Science, pages 504–512, Berlin, Germany, September 1996. Springer-Verlag.

[10] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[11] Kalmanje Krishnakumar. Micro-genetic algorithms for stationary and non-stationary function optimization. In *SPIE Proceedings: Intelligent Control and Adaptive Systems*, volume 1196, pages 289–296, 1989.

[12] Frank Kursawe. A variant of evolution strategies for vector optimization. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 193–197, Berlin, Germany, oct 1991. Springer-Verlag.

[13] Jon Rowe, Kevin Vinsen, and Nick Marvin. Parallel GAs for Multiobjective Functions. In Jarmo T. Alander, editor, *Proceedings of the Second Nordic Workshop on Genetic Algorithms and Their Applications (2NWGA)*, pages 61–70, Vaasa, Finland, August 1996. University of Vaasa.

[14] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1984.

[15] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, fall 1994.

[16] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation*, 8(2):125–147, 2000.

[17] Fengchao Xiao and Hatsuo Yabe. Microwave Imaging of Perfectly Conducting Cylinders from Real Data by Micro Genetic Algorithm Coupled with Deterministic Method. *IEICE Transactions on Electronics*, E81-C(12):1784–1792, December 1998.

[18] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization

David W. Corne, Nick R. Jerram, Joshua D. Knowles, Martin J. Oates
Department of Computer Science, University of Reading, UK
{D.W.Corne, N.R.Jerram, J.D.Knowles}@reading.ac.uk, moates@btinternet.com
FAX: +44(0) 118 975 1994, TEL: +44 (0) 118 931 8983

Abstract

We describe a new selection technique for evolutionary multiobjective optimization algorithms in which the unit of selection is a hyperbox in objective space. In this technique, instead of assigning a selective fitness to an individual, selective fitness is assigned to the hyperboxes in objective space which are currently occupied by at least one individual in the current approximation to the Pareto frontier. A hyperbox is thereby selected, and the resulting selected individual is randomly chosen from this hyperbox. This method of selection is shown to be more sensitive to ensuring a good spread of development along the Pareto frontier than individual-based selection. The method is implemented in a modern multiobjective evolutionary algorithm, and performance is tested by using Deb's test suite of 'T' functions with varying properties. The new selection technique is found to give significantly superior results to the other methods compared, namely PAES, PESA, and SPEA; each is a modern multiobjective optimization algorithm previously found to outperform earlier approaches on various problems.

1 Introduction

Standing on the shoulders of seminal research and development in the area of multiobjective evolutionary algorithms (MOEAs), such as the Niched Pareto Genetic Algorithm (Horn et al., 1994; Horn and Nafpliotis, 1994), and the Non-Dominated Sorting method (Srinivas and Deb, 1994), the last half-decade has seen an explosion of interest and development of more capable MOEAs. The techniques that have recently emerged seem to provide fast and effective approximations to the Pareto frontier for a variety of benchmark problems. These new methods include, among others, SPEA (Strength Pareto Evolutionary Algorithm –Zitzler and Thiele, 1999), PAES (Pareto Archived Evolution Strategy – Knowles and Corne, 2000), M-PAES (Memetic PAES – Knowles and Corne, 2000a), PESA (Pareto Envelope based Selection – Corne et al, 2000), MOMGA (Multi-Objective Messy Genetic Algorithm, Van Veldhuizen and Lamont, 2000), and NSGA-II (Non-Dominated Sorting genetic Algorithm II – Deb et al, 2000).

PAES, PESA, SPEA, and NSGA-II can each be considered to be 'basic' MOEAs in the sense that their flow of control is essentially a pure evolutionary algorithm framework, while the differences between them amount to explorations of various different ways to do selection and population maintenance in multiobjective spaces. Methods such as M-PAES and MOMGA, on the other hand, are more sophisticated algorithm designs in which a pure evolutionary framework is eschewed in favour of a hybrid or multi-stage flow of control. M-PAES, for example, is a memetic algorithm in which population based search is hybridised with local search, while MOMGA is a messy genetic algorithm (Goldberg et al, 1991) adapted for use in multiobjective search.

In this paper we are interested in the 'basic' evolutionary multiobjective framework, and will therefore no longer consider M-PAES, MOMGA, and other such methods, but the technique developed may of course be incorporated in sophisticated MOEAs such as M-PAES and MOMGA in future work. We describe a variation on how to do selection in basic MOEAs, and compare an MOEA which uses this technique to each of PAES, PESA and SPEA on a variety of test problems. We have not yet compared with NSGA-II, which is an omission we hope soon to rectify.

We should also mention that much impressive multiobjective optimisation work is being done in the fields of multiple criteria decision making (MCDM) and operations research. Until recently, there has been little crosstalk between these communities and the MOEA community. Strong-performing algorithms emerging from these areas include a variety of local-search based multiobjective techniques, e.g. Czyzak and Jaszkiewicz (1998), Gandibleux et al. (1996), and Hansen (1996; 1997). Comparison of such methods with modern MOEAs has been done little so far, although recent work by Zitzler and Thiele (1999) and Knowles and Corne (2000) indicate that methods such as PAES and SPEA are at least comparable in quality to these other methods.

The remainder of this paper is set out as follows. In section 2 we briefly review selection schemes in modern evolutionary multiobjective algorithms, and introduce the simple concept of region-based selection. Some simple analysis is done to suggest why region-based selection may be favoured over other methods, in terms of its maintained strong bias towards developing isolated regions of the Pareto front. In section 3 we note the algorithms and describe the test functions used in later experiments. These experiments are described in section 4 and their results are presented in section 5, and we have a concluding discussion in section 6.

2 Region-Based Selection in Evolutionary Multiobjective Algorithms

2.1 Individual-Based Selection

We will use Figure 1 to illustrate the main selection schemes used in current multiobjective evolutionary algorithms. In the figure, a number of points are plotted in objective space for a supposed two-objective problem, and we imagine that the goal is to minimize along both axes (in the directions shown by the arrows). Objective space is divided into squares (generally, 'hyperboxes' in higher dimensional objective spaces). In both PAES and PESA, the algorithms incorporate a subdivision of the objective space into hyperboxes as shown in the figure. In PESA, information concerning the occupation of hyperboxes is used for selection as follows. An archive is maintained containing only non-dominated solutions, and as such represents the algorithm's current approximation to the Pareto frontier. Selection is only from this archive. The selective fitness of an individual is simply the number of other solutions which occupy the same hyperbox as that individual. This is called the 'squeeze factor'. Tournament selection (or any other basic selection scheme) can then be used to select parents with a bias towards small squeeze factors.

In PAES, selection is rather a different affair since PAES is essentially a local search method. There is just one current solution at any time, and this is therefore always selected to be the parent of a mutant. However, when the mutant and current solution are non-dominated, a decision has to be made as regards which will become the new current solution (which can be seen indirectly as selecting the parent for the next iteration). The full details of this decision are in Knowles and Corne (2000) but for present purposes we note that, like PESA, it makes use of hyperbox occupancy.

Selection in SPEA is done via a 'Strength Pareto' scheme developed by Zitzler and Thiele (1999). This is a way of assigning selective fitness to an individual based on the number of individuals in the population which it covers – an individual *covers* another if it dominates it, or is equal to it. This method therefore relies on having population members around which are not in the current approximation to the Pareto front. In SPEA, this is organised by having two populations, an *internal* and *external* population. The external population only contains non-dominated individuals, while the internal population contains the latest crop of children produced via genetic operators, and as such may contain individuals which are dominated by members of the external population. Figure 1, may represent the combined populations at a snapshot in a run of SPEA. The point labelled X is nondominated, and hence in SPEA's external population, and it dominates two members of the external population (those contained in the region enclosed by the lines emanating from X. The Strength measure for a nondominated individual is just the number of individuals in the internal population which it covers. Strength measures for members of the internal population are derived by summing the strengths of the external population individuals which cover them. Selection is biased towards minimising this strength figure, thus preferring the exploration of less populated regions of the objective space. So, in Figure 1, Y will have a better selective strength than X.

Finally, NSGA-II uses a rather different selection technique which has been found to be both highly efficient and to perform very favourably in comparison to others. In NSGA-II, a selective fitness measure is derived for an individual by first finding the distance to the closest other individual to it for each objective in turn. The product of these distances gives a hypervolume which in turn estimates the isolation of this individ-

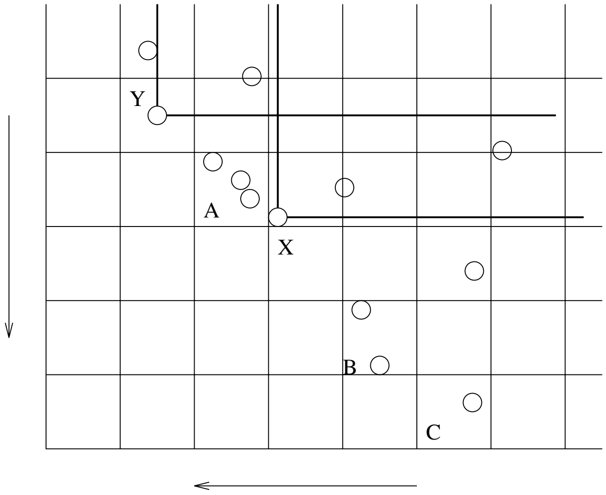


Figure 1: Illustration of Selection Methods in Modern MOEAs.

ual. Selection is therefore biased towards individuals with a high isolation value. In Figure 1, for example, the points in box A would have a low isolation value, but that of point Y would be relatively high.

Each of the selection techniques is oriented towards maintaining development of the Pareto front in a well spread manner. That is, by biasing search in the region of relatively lonely regions of the current approximation to the Pareto front, the aim is to promote an even spread of individuals along it. The main difference between the methods is the precise way in which the degree of isolation of an individual is estimated. PAES and PESA use hyperbox counts, NSGA-II uses distance to nearby individuals, and SPEA uses a somewhat indirect method which estimates an individual's isolation based on how many previously generated individuals it covers. An aspect which all of these methods share is that selection is *individual-based.* That is, the unit of selection is an individual. The different variations can therefore be seen as imposing different distributions of selection probability on the individuals, with the goal of achieving higher probabilities for those in isolated regions than those in crowded regions.

2.2 Region-Based Selection

Region-based selection provides an alternative, in which the above goal is achieved more directly. In region-based selection, the unit of selection is now a hyperbox, rather than an individual. A selective fitness is derived for a hyperbox. Using any standard selection method, a hyperbox is therefore selected, and the resulting individual chosen for genetic operations is randomly chosen from the selected hyperbox. In Figure 1, for example, hyperbox C would have a better selective fitness than hyperbox B, which in turns would have a better selective fitness than hyperbox C.

The following simple analysis suggests why region-based selection may be favoured over an individual based scheme. Assume we are using binary tournament selection without replacement in both cases. That is, binary tournament selection is used to select an individual based on selective fitnesses, whether those selective fitness are individual-based (measures of isolation such as strength or Deb's crowded-comparison measure (Deb et al, 2000)) or hyperbox-based. It is worth first considering a pathological case in which just two hyperboxes are occupied in the current approximation to the Pareto front. One is occupied by 9 individuals, and the other by a single individual. We will also assume, which seems reasonable, that the single individual is the most isolated in respect of the typical individual based selective fitness measures we have considered.

With binary tournament selection, the chance of selecting the best individual (the most isolated one) in an individual based selection scheme will be $1-(9/10)^2 = 0.19$. The chance of selecting any one of the 9 overcrowded individuals will therefore be 0.81. This does not seem to provide suitably high bias towards development in the less-crowded region. With region-based selection, however, the units of selection are the two occupied hyperboxes. The chance of choosing the least-occupied box (and hence choosing the best individual) is $1-(1/2)^2 = 0.75$. The chance of choosing any one of the more crowded individuals is therefore 0.25. With individual based selection in this example, we are actually more likely to choose a highly non-isolated individual than the most isolated one. With region-based selection, we are three times more likely to choose the isolated than any of the non-isolated individuals.

We will now take a slightly more formal look, stepping away from the pathological case to see what may be the more typical situation. We will remain interested in the relative probabilities of choosing a most isolated individual over a most crowded individual, and will continue to assume the use of binary tournament selection. Consider an approximation to the Pareto front which has b occupied hyperboxes, with n_i individuals in box i, and P individuals altogether in occupied hyperboxes, such that $\sum_{i=1}^{b} n_i = P$. Assume now, with a slight loss of generality, that a single hyperbox j has the largest b_i and another single hyperbox has the smallest n_i. The numbers of individuals in these least and most crowded boxes will be l and m respectively.

When using individual based selection, the chance of choosing an individual from the least crowded box will be $1 - ((P - l)/P)^2$. The corresponding term for the most crowded box is simply $(m/P)^2$; the ratio of these probabilities $(2Pl - l^2)/m^2$. When m is high with respect to l, the relative chance of choosing an isolated individual rather than a crowded one reduces fairly sharply, this would seem to unreasonably draw selective attention towards the crowded regions. In contrast, the corresponding ratio for region-based selection turns out to be $2b - 1$. It is unaffected by the relative numbers of individuals in the different boxes, and never less than 1 (in fact, always at least 3 when more than 1 hyperbox are occupied).

It might be thought that the same effect – that is, duly high attention to isolated regions rather than crowded ones, could be achieved by individual-based selection with a higher tournament size. However, notice that the chance of choosing an individual from the most crowded box in this case will be $(m/P)^k$, where k is the tournament size. When the tournament size is large, this will drop very sharply with a large population and a fairly even distribution of individuals among them. In these conditions, the chance of choosing an individual from the least crowded box would become unacceptably low, affecting the exploratory capabilities of the algorithm.

2.3 Complexity Issues

Here we briefly reflect on the complexity issues inherent in individual-based versus region-based selection schemes. In the context of multi-objective search, the issue of main interest to us here is the complexity of calculating selective fitness based on crowding in phenotype space. For simplicity, we will assume generational approaches in which a new population of size n is in every generation.

Individual-based selection requires estimates of the degree of 'isolation' of each individual. Accurate estimation of the relative isolation of the individuals in a population of size n would of course require n^2 comparison operations, where the distances between all distinct pairs are calculated. However, it has been found, in both NSGA-II and PAES, for example, that approximate estimates of isolation can be achieved more quickly, with quite adequate results. For example, the metric used to approximate isolation in NSGA-II (Deb et al., 2000) requires $\mathcal{O}(k \cdot n \log n)$ time, where k is the number of objectives.

In region-based selection using hyperboxes, the key computational concern is to caclulate a hyperbox ID for each individual. As indicated in Knowles & Corne (2000), in a k-objective problem using a grid of g^k hyperboxes, only $\mathcal{O}k \cdot n$ comparison operations need be made per generation. Efficiency is improved if g is a power of 2, but the broad order of complexity is just linear in n. A single pass through the hyperbox IDs then easily yields the selective fitness information required by either region-based or individual-based selection.

3 Algorithms and Test Functions

The algorithms we test in this paper are PAES, SPEA, PESA, and PESA-II. PAES is described in full in Knowles and Corne (2000), SPEA is described in Zitzler and Thiele (1999), and PESA is described in Corne et al (2000). PESA-II is a version of PESA which uses region-based selection. the parameter settings used are detailed in section 4.

Deb (1998) gives a procedure for designing tunable test functions for multiobjective optimisation. This technique enables the incorporation into objective space of a range of characteristics to varying degrees. These include discontinuity, concavity, non-uniformity of individuals along the Pareto front, and deception, each of which are considered by many to be the key characteristics which capable evolutionary multiobjective optimisers need to cope with.

In this paper we use six test functions designed using Deb's scheme. These are the functions $\mathcal{T}_1$-$\mathcal{T}_6$ which were used in a comparison of the performance of eight different MOEAs by Zitzler et al (1999), and in a comparison of three different algorithms by Corne et al (2000). The important characteristics of these functions are as follows. $\mathcal{T}_1$ plays the role of a baseline, simple test; it has a convex Pareto front, and no characteristics which should lead to particular difficulty; $\mathcal{T}_2$ has a non-convex Pareto front – this causes difficulties, for example, for several techniques from the operations research and MCDM communities, which attempt to iteratively optimise weighted sums of the objectives for different sets of weights, since solutions in the concave region are not optima of any such scalarisation; $\mathcal{T}_3$ has many discontinuities in the Pareto front; $\mathcal{T}_4$ is highly multimodal and has 21^9 Pareto fronts; $\mathcal{T}_5$ is a deceptive problem, and $\mathcal{T}_6$ has a non-uniformly distributed search space with solutions non-uniformly distributed along the Pareto front.

Each is a two-objective problem defined on m parameters, in which both objectives are to be minimized. In five of the problems the parameters x_i were coded as a binary string decoded such that $x_i \in [0,1]$. The remaining function ($\mathcal{T}_5$) also employed a binary chromosome but this time unitation was used to evaluate each of the parameters. We encode the functions here in precisely the same way as done in Zitzler et al (1999) and Corne et al (2000).

To briefly summarize Zitzler's study, SPEA seemed to be the best algorithm overall of the eight tested. Those compared included several of the classic methods such as the Niched Pareto Genetic Algorithm (Horn et al., 1994; Horn and Nafpliotis, 1994), the Non-Dominated Sorting method (Srinivas and Deb, 1994), and various versions of SPEA. Later, in Corne et al's study (2000), SPEA was compared with PESA and PAES. PESA was found to be best overall, although on $\mathcal{T}_5$, the deceptive problem, SPEA was slightly, but certainly, the best of the algorithms compared.

4 Experimental Design

4.1 Experiments

Our experiments sought to determine the relative quality of PESA, SPEA, PAES and PESA-II, a version of PESA which incorporates region-based selection, on the Deb test functions. Parameter settings are given in Table 1.

In the next section we summarise the statistical comparison method used to analyse the results within a set of experiments.

4.2 Statistics

Given the results of several trial runs for each algorithm, we compare the performance of two or more multiobjective optimisers using a method proposed originally by Fonseca and Fleming (1995a) which we have implemented with certain extensions. When comparing two multiobjective algorithms (A and B), this method returns two numbers: the percentage of the Pareto frontier on which A conclusively beats B (based on a Mann-Whitney U test at the 95% confidence level), and the percentage of the Pareto frontier on which B beats algorithm A. For example, two well-matched MOEAs might yield a result like [3.7, 4.1], indicating that each algorithm was definitely better than the other in small regions of the space, but they performed similarly well on the majority of the Pareto frontier. A clear indication that one algorithm is superior to another, however, is given by a comparison result such as [68.3, 2.2], or [100, 0.0].

Crossover rate	0.7 in PESA, PESA-II and SPEA; not used in PAES
Crossover method	uniform in PESA, PESA-II and SPEA; not used in PAES
Mutation rate	bit-flip rate set to $1/L$ where L is chromosome length
Populations	archive 100 in all algorithms, IP size 10 in PESA and SPEA
Chromosome lengths	900 in $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$, 300 in $\mathcal{T}_4$ and $\mathcal{T}_6$, 80 in $\mathcal{T}_5$
Hyper-grid size	32×32 grid in PESA, PESA-II and PAES, not used in SPEA

Table 1: Parameter settings

In a comparison of more than two algorithms, the comparison code performs pairwise statistical comparisons, as before, for each distinct pair of algorithms. The results then show, for each algorithm, on what percentage of the discovered Pareto frontier we can be confident that it was unbeaten by any of the others, and on what percentage of the space it beat *all* of the others. For example, in Table 2, we can see that, on problem $\mathcal{T}_2$, PESA-II was unbeaten by any of the other algorithms individually on the entire Pareto surface, and conclusively superior to *all* of the others on 27.4% of the Pareto Tradeoff surface.

5 Results and Discussion

Table 2 summarises all results for the set of experiments in which each trial run was allowed 5,000 fitness evaluations. The best performing algorithm for each problem has its table entries highlighted in bold; when there is little difference between the best two (or all three), each such entry are highlighted in bold.

There are two rows for each problem; the first give the *unbeaten* statistic for each algorithm, and the second gives the *beats all* statistic. For example, on problem $\mathcal{T}_{/}$ PESA-II was unbeaten by any of the other algorithms on 100% of the combined Pareto frontier discovered over all trials, and on this frontier it was significantly superior to *all* of the others on 12.5% of it. In the case of $\mathcal{T}_{/}$ PESA-II was so much better than the other methods, we did additional trials to 20,000 evaluations to see if the other methods could 'catch up'.

Problem	PAES	SPEA	PESA	PESA-II
$\mathcal{T}_1$	66.1	1.1	**99.8**	**100**
	0	0	0	0
$\mathcal{T}_2$	0	0	72.3	**100**
	0	0	0	27.4
$\mathcal{T}_3$	65.4	22.3	78.4	**100**
	0	0	0	0
$\mathcal{T}_4$	64.4	**100**	**100**	**99.8**
	0	0.1	0	0
$\mathcal{T}_5$	0	**100**	98.6	**99.7**
	0	0	0	0
$\mathcal{T}_6$	16.7	74.5	18.8	**100**
	0	0	0	12.5
$\mathcal{T}_6$-long	2.8	0.5	1.1	**99.8**
	0.2	0	0	96.1

Table 2: Comparison of PAES, SPEA, PESA and PESA-II at 5,000 evaluations (plus an extra comparison at 20,000 evaluations for $\mathcal{T}_6$

As Table 2 shows, PESA was clearly the best method on three of the functions, and joint best with SPEA on a further two. On the one remaining function it achieved the second-best performance. SPEA is clearly best on just one function, and joint best with PESA on two. PAES is the worst performer here, being clearly worst on three of the test functions, and second or joint second best on the remaining three.

The results are summarised in Table 3, in which we show the rank for each algorithm on each problem. The rank is simply one plus the number of algorithms which clearly did better. For example, PAES has rank 3 for $\mathcal{T}_\infty$ since two algorithms (PESA and PESA-II) performed better than it on this function.

With reference to both tables 2 and 3, PESA-II clearly outperforms the other methods on the test functions examined overall. The performance on the $\mathcal{T}_{/}$is especially marked.

We will now briefly consider the differential performance in terms of the $\mathcal{T}$ problem characteristics. $\mathcal{T}_\infty$ is a straightforward problem, and we find that both PESA and PESA-II perform excellently on it, with PAES doing well too, but SPEA doing rather badly. Since the problem lacks deception, and PAES is essentially a local search procedure, the good performance of PAES, especially in comparison to SPEA, is understandable. SPEA, as hinted at in Section 2, and unlike any of the other algorithms tested here, spends significant algorithmic effort in considering non-elitist solutions. This strategy seems to be unnecessary for $\mathcal{T}_\infty$, and seems to have prevented SPEA from performing well on it in the available time. In contrast, SPEA's non-elitism is likely to be responsible for it maintaining overall best performance on the deceptive problem, $\mathcal{T}_\triangledown$, and also the highly multimodal problem, $\mathcal{T}_\triangle$. $\mathcal{T}_\in$ has a concave front; this makes it non-trivial for a local search based method and also seems to have confounded the strength pareto approach; PESA and PESA-II, especially the latter, perform very well on it. PESA-II also particularly shines on the remaining two problems, $\mathcal{T}_\ni$ and $\mathcal{T}_{/}$ which, respectively, have highly discontinuous and highly non-uniform Pareto fronts.

Problem	PAES	SPEA	PESA	PESA-II
$\mathcal{T}_1$	3	4	**1**	**1**
$\mathcal{T}_2$	3	3	2	**1**
$\mathcal{T}_3$	3	4	2	**1**
$\mathcal{T}_4$	4	**1**	**1**	**1**
$\mathcal{T}_5$	4	**1**	**1**	**1**
$\mathcal{T}_6$	4	2	3	**1**
$\mathcal{T}_6$-long	2	4	3	**1**

Table 3: Summary of PAES, SPEA, PESA and PESA-II comparisons on Functions $\mathcal{T}_1$-$\mathcal{T}_6$

6 Conclusion

We have described region-based selection as an alternative selection scheme for use in evolutionary multiobjective optimisation. We have implemented it within PESA, although it may of course be employed in most evolutionary multiobjective frameworks. Experiments on functions from Deb's test suite seem to confirm that region-based selection is a very promising technique. PESA-II, which employed the new selection method, was only beaten (and then slightly) on two of the test problems. These were the highly multimodal problem, and the deceptive problem; hence, the relatively low profile for region-based selection in these cases can potentially be explained by the fact that SPEA is non-elitist (a highly helpful feature in problems with such characteristics), while region-based selection was implemented in an entirely elitist method

(PESA).

One promising avenue for further work would seem to be the deployment of region-based selection in a non-elitist framework. However this is not trivial; considering occupied hyperboxes in dominated regions of the space requires us to have a way of preferring, for example, a hyperbox on the Pareto frontier over a dominated hyperbox which has the same number of occupants. Another complicating factor is that a hyperbox may contain both dominated and nondominated individuals. We are thinking along the lines of using the Strength Pareto approach (Zitzler and thiele, 1999) to deal with these issues. A simple alternative might be to simply do region-based selection on the nondominated frontier most of the time, but for a portion of the time select from dominated individuals based on their strength Pareto fitnesses. A further alternative would be to only use region-based selection, but apportion algorithm effort between selecting in this way from different Pareto frontiers, akin to the nondominated sorting approach (Srinivas and Deb, 1994).

There are runtime complexity issues which we have not dealt with in this paper. For example, finding occupied hyperboxes can be done quickly, though is not trivial. Depending on the enclosing algorithm framework, the hyperbox and related datastructures may or may not need constant updating. Also, region-based selection (and any hypergrid method) requires the choice of a parameter to define the individual hyperbox dimensions. NSGA-II (Deb et al, 2000), for example, requires no such parameter. Preliminary investigations suggest that results are not overly sensitive to the hyperbox dimension parameter, although much more investigation needs to be done to determine if this is generally the case.

Acknowledgments

The authors are grateful to British Telecommunications Plc for financial support of the authors. We also acknowledge the helpful criticism of the anonymous reviewiers.

References

Corne, D.W., Knowles, J.D. and Oates, M.J. (2000). The Pareto-Envelope based Selection Algorithm for Multiobjective Optimisation, in Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J. and Schwefel, H-P. (eds.), *Parallel Problem Solving from Nature - PPSN VI*, Springer Lecture Notes in Computer Science, pp. 869–878.

Czyzak, P. and Jaszkiewicz, A. (1998). Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis 7*, 34–47.

Deb, K. (1998). Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. Technical Report CI-49/98, Department of Computer Science, University of Dortmund.

Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T. (2000). A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II, in Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J. and Schwefel, H-P. (eds.), *Parallel Problem Solving from Nature - PPSN VI*, Springer Lecture Notes in Computer Science, pp. 849–858.

Fonseca, C. M. and Fleming, P. J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation 3*, 1–16.

Fonseca, C. M. and Fleming, P. J. (1995a). On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. In Voigt, H-M., Ebeling, W., Rechenberg, I. and Schwefel, H-P., editors, *Parallel Problem Solving From Nature - PPSN IV*, pages 584–593, Springer.

Gandibleux, X., Mezdaoui N. and Freville, A. (1996). A tabu search procedure to solve multiobjective combinatorial optimization problems. In Caballero, R. and Steuer, R., editors, in *Proceedings volume of Multiple Objective Programming and Goal Programming '96*, Springer-Verlag.

Golberg, D.E., Deb, K., Korb, B. (1991) Don't Worry, Be Messy. in *Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 24-30.*

Hansen, M. P. (1996). Tabu Search for Multiobjective Optimization : MOTS. Presented at MCDM '97, *Cape Town, South Africa, Jan 6-10.*

Hansen, M. P. (1997). Generating a Diversity of Good Solutions to a Practical Combinatorial Problem using Vectorized Simulated Annealing. Submitted to Control and Cybernetics*, August 1997.*

Horn, J., Nafpliotis, N., Goldberg, D.E. (1994). A niched Pareto genetic algorithm for multiobjective optimization, in Proceedings of the First IEEE Conference on Evolutionary Computation IEEE World Congress on Computational Intelligence, Volume 1, *pages 67–72. Piscataway, NJ: IEEE Service Centre.*

Horn, J. and Nafpliotis, N. (1994). Multiobjective Optimization Using The Niched Pareto Genetic Al-

gorithm. IlliGAL Report 93005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Champaign.

Knowles, J. D. and Corne, D. W. (1999). Local Search, Multiobjective Optimisation and the Pareto Archived Evolution Strategy, in Proceedings of the Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, pp. 209–216.*

Knowles, J. D. and Corne, D. W. (2000). Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. Evolutionary Computation*, 8(2):149–172.*

Knowles, J.D. and Corne, D. (2000a). M-PAES: A Memetic Algorithm for Multiobjective Optimisation, in Proceedings of the 2000 Congress on Evolutionary Computation*, IEEE Neural Networks Council, pp. 325-332.*

Parks, G. T., Miller, I. (1998). Selective Breeding in a Multiobjective Genetic Algorithm. In Fifth International Conference on Parallel Problem Solving from Nature (PPSN-V)*, pages 250–259. Springer.*

Srinivas, N., Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation, 2(3)*, 221–248.*

Van Veldhuizen, D.A. and Lamont, G.B. (2000). Multiobjective Optimization with Messy Genetic Algorithms, in Proceedings of the 2000 ACM Symposium on Applied Computing*, ACM Press, pages 470-476.*

Zitzler, E. and Thiele, L. (1999). Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation, 2(4)*, 257–272.*

Zitzler, E., Deb, K. and Thiele, L. (1999) Comparison of multiobjective evolutionary algorithms: empirical results. Technical report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich.

Stepwise Adaption of Weights with Refinement and Decay on Constraint Satisfaction Problems

B.G.W. Craenen
Computational Intelligence Group
Faculty of Exact Sciences
Vrije Universiteit Amsterdam
bcraenen@cs.vu.nl

A.E. Eiben
Computational Intelligence Group
Faculty of Exact Sciences
Vrije Universiteit Amsterdam
gusz@cs.vu.nl

Abstract

Adaptive fitness functions have led to very successful evolutionary algorithms (EA) for various types of constraint satisfaction problems (CSPs). In this paper we consider one particular fitness function adaptation mechanism, the so called Stepwise Adaption of Weights (SAW). We compare algorithm variants including two penalty systems and we experiment with extensions of the SAW mechanism utilizing a refinement function and a decay function. Experiments are executed on binary CSP instances generated by a recently proposed method (method E). This new method for generating problem instances allows one single hardness parameter and is well suited to study algorithmic behavior around the phase transition. The results show that the original version of the SAW mechanism is very robust and has a comparable or better performance than the extended SAW mechanisms.

1 INTRODUCTION

Informally, a constraint satisfaction problem (CSP) consists of finding an assignment of values to variables in such a way that the restrictions imposed by the given set of constraints are satisfied. Constraint satisfaction is a fundamental topic in artificial intelligence with great practical and theoretical relevance. On the practical side, CSPs have relevant applications in planning, default reasoning, scheduling, etc. and a great deal of practical problems are constrained. Theoretically, CSPs are, in general, computationally intractable (NP-hard) thereby forming a big challenge to algorithm designers.

Evolutionary algorithms are known for their good performance in the field of optimization. Constraint handling, however, is not straightforward in an EA as the traditional search operators, mutation and recombination do not heed constraints (or their violation). Nevertheless, there is a growing body of literature on applying EAs to various CSPs, such as graph-coloring, satisfiability, or randomly generated binary CSPs. One research line is based on a pure penalty approach, where the evolutionary algorithm handles all constraints indirectly. That is, *all* constraint violations are turned into penalties and the EA is "only" optimizing an unconstrained problem where the fitness function is composed from these penalties. In this approach any direct constraint handling (e.g. specific constraint respecting mutation or crossover operators, or repair mechanisms fixing some constraint violations) is absent, making it very transparent and general.

The SAW procedure is an add-on to this general scheme to boost performance (to minimize the penalties more effectively): It adaptively changes the composition of the fitness function during the search process. Here we experiment with the SAW EA with two penalty systems (penalizing the violated constraints vs. penalizing the wrongly instantiated variables), different refinement functions, and decay functions, so as to give a complete overview of different variations on this algorithm.

Another novelty in the present work is the usage of a recently proposed problem instance generator. As shown by Achlioptas et al. in [1], the widely applied CSP generators (much used in EA research) have serious deficiencies. Most importantly, the generated instances tend to be asymptotically unsolvable, preventing a sound study of algorithmic behavior around the phase transition. The proposed alternative cures this problem and has an additional nice feature: It allows one single hardness parameter, making the presentation (and interpretation) of results much easier.

The paper is organized as follows. Section 2 will define the notation and terminology we use for describing CSPs and in section 4 we will examine further how CSPs are generated using model E. In section 4.1 we will discuss how the standard stepwise adaption of weights method will work on these CSPs while in section 4.2 the refining function and the adaptation mechanisms are discussed. In section 4.3 the decay mechanisms are discussed, followed by the experimental results in section 5. Conclusions are given in section 6.

2 NOTATION AND TERMINOLOGY

A *constraint network* consists of a set of variables $X_1, \ldots, X_n$ with respective domains $D_1, \ldots, D_n$, and a set of constraints $\mathcal{C}$. The Cartesian product of sets $D_1 \times \cdots \times D_n$ is called the *search space* and denoted by S. For $2 \leq k \leq n$, a constraint $c_{j_1,\ldots,j_k} \in \mathcal{C}, j = 1, \ldots, m$ is a subset of $D_{j_1} \times \cdots \times D_{j_k}$, where the $j_1, \ldots, j_k$ are distinct. We say that $c_{j_1,\ldots,j_k}$ is of arity k and that it bounds the variables $X_{j_1}, \ldots, X_{j_k}$ and that $\mathcal{C}^{j_1,\ldots,j_k}$ is the set of constraints that bound variables $X_1, \ldots, X_k$[1] For a given constraint network, the *Constraint Satisfaction Problem* (CSP) asks for all the n-tuples $(d_1, \ldots, d_n)$ of values such that $d_i \in D_i$, $i = 1, \ldots, n$, and for every $c_{j_1,\ldots,j_k} \in \mathcal{C}$, $(d_{j_1}, \ldots, d_{j_k}) \notin c_{j_1,\ldots,j_k}, j = 1, \ldots, m$. Such an n-tuple $s \in S$ is called a *solution* of the CSP. The decision version of the CSP is determining if a solution exists.

For an instance Π of CSP with n variables, its *constraint hypergraph* G^{Π} has n vertices $v_1, \ldots, v_n$, which correspond to the variables of Π and it contains a hyperedge $\{v_{j_1}, \ldots, v_{j_k}\}$ if and only if there exists a constraint of arity k that bounds the variables $X_{j_1}, \ldots, X_{j_k}$. The following convenient graph-theoretic representation of a CSP instance Π will be used; the incompatibility hypergraph of Π, C^{Π}, is an n-partite hypergraph of which the ith part corresponds to variable X_i of Π which has exactly $|D_i|$ vertices, one for each value in D_i. In C^{Π} there exists a hyperedge $\{v_{j_1}, \ldots, v_{j_k}\}$, if and only if the corresponding values $d_{j_1} \in D_{j_1}$, $d_{j_2} \in D_{j_2}, \ldots, d_{j_k} \in D_{j_k}$ are in (not allowed by) some constraint that bounds the corresponding variables. Hence, the decision version of CSP is equivalent to asking if there exists a set of vertices in C containing exactly one vertex from each part while not 'containing' any hyperedge, i.e., if there exists an independent set with one vertex from each part[2]. We define the Boolean function ϕ on the search space S as the *feasibility condition*, with $\phi(s) = true$ if and only if s is a solution of S, while the set $\{s \in S | \phi(s) = true\}$ will be called the *feasible search space*.

Note that for the sake of simplicity we study binary CSPs where all constraints have arity $k = 2$ (bound two variables) and where all the variable domains contain the same number of values D. We adhere to this simplification because it restricts experimental complexity and every CSP of arity larger than two has an equivalent binary CSP ([17]).

3 GENERATING CSPS

In [1], Achlioptas et al. show that the so-called Models A to D are unsuitable for the study of phase transition and threshold phenomena such as CSPs. This is because the instances they asymptotically generate have almost certainly no solutions.

A general framework for these models, presented in [15, 16] works in two steps:

Step 1 *Either (i) each one of the* $\binom{n}{2}$ *edges is selected to be in G independently of all other edges with probability* p_1 *(constraint density), or (ii) we uniformly select a random set of edges of size* $p_1\binom{n}{2}$.

Step 2 *Either (i) for every edge of G each one of the* D^2 *edges in C is selected with probability* p_2 *(constraint tightness), or (ii) for every edge of G we uniformly select a random set of edges in C of size* $p_2 D^2$

Combining the options for the two sets, we get four slightly different models for generating random CSPs, in particular, in the terminology used in [16], if both Step 1 and 2 are done with option (i), we get Model A, while if both steps are done with option (ii), we get Model B.

As Achlioptas et al. show in [1] Model A generates almost certainly unsatisfiable instances for every $p_2 \neq 0$, while Model B generates almost certainly unsatisfiable instances for every $p_2 \geq 1/D$ (analogously for the other two models). They further show that one source of this asymptotic insolubility is the appearance of 'flawed' values, i.e., values that are incompatible with all the values of some other variable. A number of experimental studies, as reported in [14] have avoided this pitfall, but many others did not.

---

[1] We use shorthand c_j and $\mathcal{C}^j$ if it cannot lead to confusion.

[2] The superscript from both the constraint and the incompatibility hypergraph will be omitted when it is clear from the context what instance is referred too

In the same paper, Achlioptas at al., proposed an alternative model for generating random CSP instances (Model E), which does not suffer from the deficiencies underlying the other models. This model resembles the model used for generating random boolean formulas for the satisfiability problem and the constraints it generates are similar to the 'nogoods' proposed by Williams and Hogg ([18]). This model is defined as:

Definition 1 *C^{Π} is a random n-partite graph with D vertices in each part constructed by uniformly, independently and with repetitions selecting $m = p\binom{n}{k}D^k$ hyperedges out of the $\binom{n}{k}D^k$ possible ones, with $k = 2$ for binary constraint networks. Also, let $r = m/n$ denote the ratio of the selected edges to the number of variables.*

Such a model can be fully specified as $\mathrm{E}(n, m, D, k)$, where n is the number of variables, m is the number of constraints, D is the number of values in each domain and k is the arity of each constraint. Informally one could say that Model E works by choosing uniformly, independently and with repetitions conflicts between two values of two different variables. The paper continues by stating that for a random instance Π generated using Model E, if we have $r < 1/2$, Π almost certainly has a solution and it is possible to bound the underconstrained and overconstrained regions.

It was known for Model A to D, that, when one of their parameters was varied, the generated CSP would exhibit a so called *phase transition*, where problems change from being relatively easy to solve to being very easy to prove unsolvable. The region where the probability that a problem is soluble changes from almost zero to almost one is generally indicated as the *mushy region*. In the mushy region, problems are in general difficult to solve or prove unsolvable and therefore of particular interest when comparing different algorithms for efficiency. In [1] Achlioptas et al. show that Model E also exhibits a phase transition when one of its variables is changed and, they give bounding formulas for the mushy region. In this paper, all CSP instances are generated using Model E with $n = 15$ variables, domain size $D = 15$, $k = 2$, probabilities from the set $\{0.20, 0.22, 0.24, 0.26, 0.28, 0.30, 0.32, 0.34, 0.36, 0.38\}$, and the corresponding values of m (see Definition 1). This puts all generated instances between the under- and overconstrained regions.

4 ADAPTIVE FITNESS FUNCTIONS FOR CSPS

4.1 STANDARD STEPWISE ADAPTION OF WEIGHTS

The Stepwise Adaptation of Weights (SAW) mechanism has been introduced by Eiben and van der Hauw [7, 8]. In several comparisons the SAW EA proved to be a superior technique for solving specific CSPs [9, 2]. The basic idea behind the SAW mechanism is that constraints that are not satisfied after a certain number of search steps (i.e. fitness evaluations), must be hard and therefore be given more attention. This is realized by using a weighted sum of constraint violations as fitness function and varying these weights to direct the search. Technically, all weights are given an initial value of 1 and re-setting them happens by adding a value Δw after a certain predefined number of evaluations. The best individual of the given population is used as reference for weight updates. Constraints that are violated in the current-best-individual are given a higher weight during an update operation.

The two penalty systems we compare here differ in the elementary penalty terms the fitness function is composed from. Namely, these terms can be based on:

1. *constraints* that are violated, or on
2. *variables* that are wrongly instantiated.

These two mechanisms can formally be described as follows:

$$f_1(s) = \sum_{i=1}^{m} w_i \cdot \chi(s, c_i), \tag{1}$$

where

$$\chi(s, c_i) = \begin{cases} 1 & \text{if } s \text{ violates } c_i \\ 0 & \text{otherwise} \end{cases}$$

respectively

$$f_2(s) = \sum_{i=1}^{n} w_i \cdot \chi(s, \mathcal{C}^i), \tag{2}$$

where

$$\chi(s, \mathcal{C}^i) = \begin{cases} 1 & \text{if } s \text{ violates at least one } c \in \mathcal{C}^i \\ 0 & \text{otherwise} \end{cases}$$

Obviously, for the above functions f_1, f_2 and for each $s \in S$ we have that $\phi(s) = \mathit{true}$ if and only if $f_i(s) = 0$ with $i \in \{1, 2\}$.

The corresponding adaption schemes, that is, weight update mechanisms, are as follows:

$$w_i \leftarrow w_i + \chi(X^*, c_i) \text{ for } i \in \{1, \dots, m\} \ (\mathrm{SAW}_{con})$$

respectively

$$w_i \leftarrow w_i + \chi(X^*, \mathcal{C}^i) \text{ for } i \in \{1, \ldots, n\} \text{ (SAW}_{var}\text{)}$$

with X^* denoting a variable in the best individual in the population found so far.

4.2 REFINING FUNCTIONS

In [12] and [13], Gottlieb and Voss have shown that refining functions improve the performance of SAW for 3-SAT problems. Here we investigate if using these refining functions also improves performance on randomly generated binary CSPs. Extending the SAW EA with a refining function means to add a term $\alpha \cdot r(X)$ to the fitness functions leading to the following definitions:

$$f_3(s) = \sum_{i=1}^{m} w_i \cdot \chi(s, c_i) + \alpha \cdot r(X) \qquad (3)$$

respectively

$$f_4(s) = \sum_{i=1}^{n} w_i \cdot \chi(s, \mathcal{C}^i) + \alpha \cdot r(X) \qquad (4)$$

Adding $\alpha \cdot r(X)$ to the fitness function makes it possible to differentiate between individuals having the same basic fitness value. Note that the refining function values are limited to the range $[0, 1)$. By using a refining factor α, the influence of the refining function on the fitness function can be tuned. The original definition of the refining function used in [13] is adapted to CSPs as follows:

$$r(X) = \frac{1}{2}\left(1 + \frac{\sum_{j=1}^{n} K(X_j) \cdot v_j}{1 + \sum_{j=1}^{n} |v_j|}\right)$$

with

$$K(X_j) = \begin{cases} 1 & \text{if } X_j \text{ is not causing a violation in } X^* \\ -1 & \text{otherwise} \end{cases}$$

where weight v_j belongs to variable X_j. Note that $r(X)$ always adds a term concerning the wrongly instantiated variables and we get two separate sets of weights. In case of f_1 (f_3) we get weights w_i with $i \in \{1, \ldots, m\}$ for the constraints and weights v_j with $j \in \{1, \ldots, n\}$ for the variables. In case of f_2 (f_4) both sets of weights w_i with $i \in \{1, \ldots, n\}$ and v_j with $j \in \{1, \ldots, n\}$ concern the variables.

In both cases the update rule SAW$_{con}$, respectively SAW$_{var}$ can be used for the w's and we introduce a new rule for the v's. The values for the v's are also initiated with 1 and updated simultaneously with the w's.

Following [13] we, in fact, introduce two different update rules for the weights in the refinement function. The rule AW1 is a problem-independent version which reflects a moderate adjustment of the weights towards the complement of the current best individual X^*,

$$v_j \leftarrow v_j - K(x_j^*) \quad \text{for } j \in \{1, \ldots, n\} \quad \text{(AW1)}$$

The rule AW2 is a problem-dependent version that uses CSP specific knowledge in $\mathcal{C}^l$:

$$v_j \leftarrow v_j - \sum_{l=1}^{n} K(x_l^*) |\mathcal{C}^l(x_l^*)| \quad j \in \{1, \ldots, n\} \quad \text{(AW2)}$$

AW2 takes into account that it is necessary to change a variable that has an unsatisfied constraint that binds it in order to improve the current solution and hence guides the EA towards solutions satisfying yet unsatisfied constraints. We denote the SAW algorithm that uses f_3 with update rule $AW1$ as SAW$_{con,ref,AW1}$ and if it uses f_4 with update rule $AW1$ we use SAW$_{var,ref,AW1}$. We replace $AW1$ subscript with $AW2$ if the SAW algorithm uses the $AW2$ update rule.

4.3 DECAY

In [11], Frank showed that WGSAT, a local search algorithm using clause weights, is susceptible to large absolute weights and convergence of relative weights. To overcome this problem he suggested a decay factor. A decay factor yields a chance to reduce high absolute weights, which allows a correction of inappropriately adapted weights. Given the decay factor $\beta \in [0, 1]$, we consider the decayed adaption schemes:

$$\begin{array}{ll} w_i \leftarrow \beta w_i + \chi(X^*, c_i) & (\text{SAW}_{var,d}) \\ w_i \leftarrow \beta w_i + \chi(X^*, \mathcal{C}^i) & (\text{SAW}_{con,d}) \\ v_j \leftarrow \beta v_j - K(x_j^*) & (\text{AW1}_d) \\ v_j \leftarrow \beta v_j - \sum_{l=1}^{n} K(x_l^*) |\mathcal{C}^l(x_l^*)| & (\text{AW2}_d) \end{array}$$

As already observed for WGSAT in [11] and for SAW in [13], for 3-SAT, the good β-values are very close to 1. The same behavior for large decay rates occurred using the SAW mechanisms for CSPs as it did for 3-SAT. If the decay rate was too large (β too small, approximately $\beta \leq 0.9$), it destroyed much of the information learned during the search process.

5 EXPERIMENTAL RESULTS

We used a steady-state evolutionary algorithm with order-based representation that proved to be the best option in [10]. Here an individual is a permutation of the variables and a decoder is used to assign domain values to each variable in the order they appear in a given permutation. The decoder sequentially takes variables from the permutation and tries to instantiate it with values that do not violate any constraints that bind already instantiated variables. If the decoder does not find such a value, the variable is left uninstantiated. (Technically, it is instantiated to a special value indicating a conflict.) Earlier work has shown that small populations produce the best performance and that for CSPs a population size of just 1 is optimal. Therefore, we use a $(1+1)$ style EA and no crossover operation is needed. The mutation operator is a simple swap operator, which randomly chooses one variable in the given permutation and swaps it with another randomly chosen variable. The initial population is generated randomly and the weights are adapted each time 250 evaluations have been done. (Preliminary experiments with different adaption periods showed little differences in performance; 250 gave just slightly better results than other values.) After a maximum of 100,000 evaluations, the runs were terminated. As mentioned in section 3, we used a set of 10 different probabilities for the Model E CSP generator. With each of these probabilities we generated 25 instances and performed 10 runs over each instance, resulting in 250 runs for every p value for each algorithm variant.

We used two measures of comparison for the algorithms; first Success Rate (SR), which denotes the percentage of the runs that were completed with a solution[3]; second, Average number of Evaluations to a Solution (AES). Note that the last measure is only defined when a solution was found and that, although it seems a 'fair' measure, it could be misleading as some EAs use 'hidden labor' which could be invisible to the measure. An example of hidden labor could be some of the work done in $(AW2)$, where problem-specific knowledge was used in the fitness function.

We experimented with SAW algorithms using the different fitness functions (f_1, f_2, f_3, f_4), using the two refining functions ($AW1$ and $AW2$) for fitness functions f_3 and f_4 and using the decay mechanisms.

The results are depicted in figure 1, the corresponding numerical figures are given in table 1. These outcomes show that there is little difference between the SAW algorithms that consider either constraints or variables (fitness functions f_1 or f_2). It might be observed that SAW$_{var}$ has a small advantage in Average Evaluations to Solution (AES) but when comparing Success Rate (SR), the differences are small. In instances with probability 0.34 and 0.38 SAW$_{var}$ has a better SR, while in instances with probability 0.36 SAW$_{con}$ solved more instances. Other instances were solved by both algorithms equally well (SR) and until $p = 0.24$ the speed figures (AES) are also the same. The standard deviations of the averages for AES (not presented here) were also so close that no further distinction could be made.

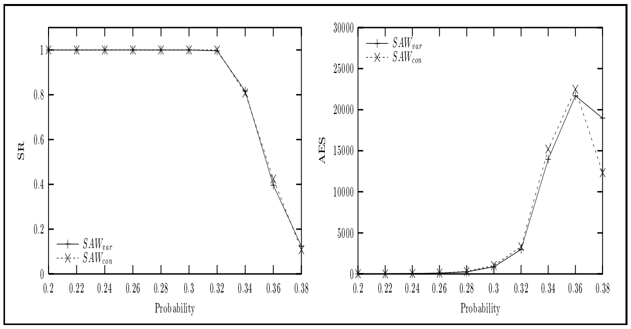


Figure 1: SR and AES graphs for SAW$_{var}$ and SAW$_{con}$

The little difference between trying to solve CSPs with either penalizing variables or constraints is somewhat surprising if we consider that there are much more constraints than variables. This implies that SAW$_{con}$ working with the fitness function f_1 has more information, but apparently f_2 is already "strong" enough.

	SAW$_{var}$		SAW$_{con}$	
	SR	AES	SR	AES
0.20	1	9.936	1	9.936
0.22	1	17.304	1	17.304
0.24	1	45.632	1	45.632
0.26	1	104.448	1	106.336
0.28	1	258.28	1	311.068
0.30	1	870.556	1	1023.66
0.32	1	2984.63	1	3322.18
0.34	0.816	13962.6	0.808	15212.3
0.36	0.396	21683.3	0.424	22480.3
0.38	0.124	18968.3	0.108	12332.2

Table 1: Numerical results for SAW$_{var}$ and SAW$_{con}$

The results of the experiments with the extended SAW mechanism are given in table 2 for SAW$_{var,ref,AW1}$ and SAW$_{con,ref,AW1}$ and in table 3 for SAW$_{var,ref,AW2}$ and SAW$_{con,ref,AW2}$. The figures show that the addition of the refining function AW1 produced no improvement at all. Experiments were also performed with varying refinement factors, even up to values where the performance of the algorithms

---

[3] We use the decimal notation of a percentage: 10% = 0.10

began to deteriorate. Differences between the refining functions (AW1 and AW2) are also small, which indicates that adding extra domain information, in the form of $\mathcal{C}^l$, also did not improve search performance. This is also surprising because one would expect to increase search speed when incorporating extra domain knowledge. This might come at the cost of premature convergence because of searching too greedily

	SAW$_{var,ref,AW1}$		SAW$_{con,ref,AW1}$	
	SR	AES	SR	AES
0.20	1	9.936	1	9.936
0.22	1	17.304	1	17.304
0.24	1	45.632	1	45.632
0.26	1	104.448	1	106.336
0.28	1	258.28	1	311.068
0.30	1	870.556	1	1023.66
0.32	1	2984.63	1	3322.18
0.34	0.816	13962.6	0.808	15212.3
0.36	0.396	21683.3	0.424	22480.3
0.38	0.124	18968.3	0.108	12332.3

Table 2: Results for SAW$_{var,ref,AW1}$ and SAW$_{con,ref,AW1}$

	SAW$_{var,ref,AW2}$		SAW$_{con,ref,AW2}$	
	SR	AES	SR	AES
0.20	1	9.936	1	9.936
0.22	1	17.304	1	17.304
0.24	1	45.632	1	45.632
0.26	1	104.448	1	104.524
0.28	1	258.28	1	252.996
0.30	1	870.556	1	1048.48
0.32	0.996	2984.63	1	3430.64
0.34	0.816	13962.6	0.840	16487.5
0.36	0.396	21683.3	0.368	20957.4
0.38	0.124	18968.3	0.116	23669.4

Table 3: Results for SAW$_{var,ref,AW2}$ and SAW$_{con,ref,AW2}$

When interpreting these results, recall that the refining function was designed to distinguish between individuals having the same basic fitness value. The lack of improvement when using refining functions seems to imply that in case of binary CSPs, the basic functions f_1 and f_2 contain sufficient information to guide the search successfully.

Experiments using decay factor for solving SAT problems showed that values around $\beta = 1$ work best, cf. [11, 13]. Our studies with random binary CSPs also indicated the same. However, we also found that applying decay to the SAW technique did not change algorithm performance significantly when β values were taken from the set $\{0.95, 0.96, 0.97, 0.98, 0.99, 1.00\}$. Note that $\beta = 1.00$ amounts to no decay. Therefore this observation implies that algorithm variants without decay and with decay using a good β value are not performing differently.

Figure 2 gives an illustration by showing typical runs using different decay factors. These runs were performed on instances generated with probability 0.32. Note that the figure only presents the search speed results (AES). The success rates are not given because all algorithms solved all instances, except for SAW$_{var,ref,AW1,d}$ with decay factor 0.98 which solved 096% of the instances (a SR of = 0.96). The curves indicate a difference between SAW$_{var,ref}$ and the other algorithms. Namely, SAW$_{var,ref}$ using either AW1 or AW2 seems to improve when switching off decay, i.e. for $\beta = 1.0$. For all other algorithms a decay factor has no significant influence on the search speed. Recall that the decay of weights in the SAW fitness function was added to actively suppress the growth of weight values. These results indicate that such a growth – identified as dangerous in related fields, cf. [11, 13], – either does not occur or is not harmful in case of random binary CSPs.

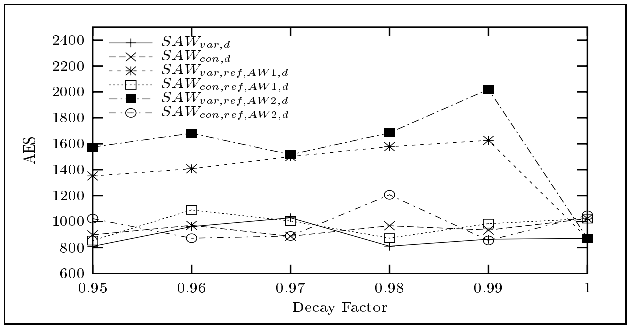


Figure 2: graphs of SAW$_{var,d}$ and SAW$_{con,d}$ with different decay factors. $p = 0.32$, $\Delta = 250$, refinement-factor $\alpha = 40$

6 CONCLUSIONS

The research presented in this paper had a twofold objective: presenting and illustrating a recently proposed problem instance generator for binary CSPs and comparing a number of variants and extensions of the SAW technique. In particular, we used a CSP generator based on the so-called model E in [1]. We found that the phase transition effects were clearly observable and by the use of one single hardness parameter the results were easier to present – and to interpret – than in case of the formerly used two-parameter-based generators [3, 4, 5, 10]. The new generator deserves a recommendation for further experimental research.

As for the algorithm variants, we experimented with the unextended version of the SAW technique with two penalty systems: one calculated over the variables and one calculated over the constraints. They turned out to yield very similar performance, which is surprising. The constraint based fitness function, after all, is based on more information as there are usually more constraints than variables and one would expect to improve a search algorithm when using more information.

Inspired by related work on satisfiability problems we have also experimented with two extensions of SAWing. In particular, we tried refining functions and a decay mechanism. A refining function $\alpha \cdot r(X)$ added to the fitness function makes it possible to differentiate between individuals having the same basic fitness value. We found that the application of refining functions did not improve performance and that varying the refining factor α did not have any influence on performance. These results seem to imply that in case of binary CSPs the basic functions f_1 and f_2 contain sufficient information to guide the search successfully.

The addition of the decay factor did not improve the performance in general either, neither for the original SAW techniques, nor for the SAW technique with a refining function. In fact it had a negative effect on $\mathrm{SAW}_{var,ref}$. This might be the consequence of the relatively small problem size, where the accumulation of large relative weights stays within limits and thus does not need a counterforce. This finding points to the same direction as our conclusion about refinement functions: the basic SAW mechanism is powerful enough to solve random binary CSPs.

All in all, the comparison of the algorithm variants shows a surprising, but pleasant picture: The simplest setup (SAW with variable related penalties, no extensions) is as good or better than any of the more sophisticated variants. Although current and future research will undoubtfully refine this picture, for the time being this is good news for algorithm designers.

Further research is carried out with new types of refining functions, problems with varying sizes (scale-up), and larger population sizes.

References

[1] D. Achlioptas, L.M. Kirousis, E. Kranakis, D. Krizanc, M.S.O. Molloy, and Y.C. Stamatiou. Random constraint satisfaction: A more accurate picture. In G. Smolka, editor, *Principles and Practice of Constraint Programming — CP97*, number 1330 in Lecture Notes in Computer Science, pages 107–120, Berlin, 1997. Springer-Verlag.

[2] Th. Bäck, A.E. Eiben, and M.E. Vink. A superior evolutionary algorithm for 3-SAT. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Proceedings of the 7th Annual Conference on Evolutionary Programming*, number 1477 in Lecture Notes in Computer Science, pages 125–136, Berlin, 1998. Springer-Verlag.

[3] J. Bowen and G. Dozier. Solving constraint satisfaction problems using a genetic/systematic search hybride that realizes when to quit. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 122–129. Morgan Kaufmann Publishers, Inc., 1995.

[4] G. Dozier, J. Bowen, and D. Bahler. Solving small and large constraint satisfaction problems using a heuristic-based microgenetic algorithm. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 306–311. IEEE Computer Society Press, 1994.

[5] G. Dozier, J. Bowen, and A. Homaifar. Solving constraint satisfaction problems using hybrid evolutionary search. *Transactions on Evolutionary Computation*, 2(1):23–33, 1998.

[6] A.E. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors. *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, number 1498 in Lecture Notes in Computer Science, Berlin, 1998. Springer-Verlag.

[7] A.E. Eiben and J.K. van der Hauw. Solving 3-SAT with adaptive Genetic Algorithms. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE Computer Society Press, 1997.

[8] A.E. Eiben and J.K. van der Hauw. Adaptive penalties for evolutionary graph-coloring. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution '97*, number 1363 in Lecture Notes in Computer Science, pages 95–106, Berlin, 1998. Springer-Verlag.

[9] A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[10] A.E. Eiben, J.I. van Hemert, E. Marchiori, and A.G. Steenbeek. Solving binary constraint satisfaction problems using evolutionary algorithms with an adaptive fitness function. In Eiben et al. [6], pages 196–205.

[11] J. Frank. Learning short-term weights for GSAT. Technical report, University of California at Davis, oct 1996.

[12] J. Gottlieb and N. Voss. Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In Eiben et al. [6].

[13] J. Gottlieb and N. Voss. Adaptive fitness functions for the satisfiability problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Proceedings of the 6th Conference on Parallel Problem Solving from Nature*, number 1917 in Lecture Notes in Computer Science, Berlin, 2000. Springer-Verlag.

[14] E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. Random constraint satisfaction: theory meets practice. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 325–339, Berlin, 1998. Springer-Verlag.

[15] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Journal of Artificial Intelligence*, 81:81–109, 1996.

[16] B.M. Smith and M.E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Journal of Artificial Intelligence*, 81(1-2):155–181, 1996.

[17] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press Limited, 1993.

[18] C.P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Journal of Artificial Intelligence*, 70:73–117, 1994.

Evolving Complex Fuzzy Classifier Rules Using a Linear Tree Genetic Representation

Dipankar Dasgupta

Division of Computer Science
Mathematical Sciences Department
The University of Memphis
Memphis, TN 38152.
ddasgupt@memphis.edu

Fabio A. González

Division of Computer Science
Mathematical Sciences Department
The University of Memphis
and Universidad Nacional de Colombia
fgonzalz@memphis.edu

Abstract

The paper proposes a linear representation of tree structures in order to evolve complex fuzzy rule sets for solving classification problems. In particular, linguistic rules are evolved, where the condition part of a rule can have an arbitrary structure of conjunctions and disjunctions. We describe an efficient rule representation scheme, which uses a genetic algorithm. The method is tested with a number of benchmark data sets and some results are reported.

1 INTRODUCTION

The problem of classification has been studied extensively in machine learning (Holte,93; Michalski,98; Weiss,91), and has recently received a lot of attention in the emerging area of Data Mining (or Knowledge Discovering) (Han,00; Michalski,98). The problem of classification can be stated as follows: Given a set of classified elements (training set), build a system (classifier) that is capable of categorizing unlabeled elements (testing set) where the label of an element represents the class to which it belongs.

There exist many approaches for solving classification problems (Weiss,91): statistical methods, decision trees, neural networks, rule-based methods, etc.; and all of them have some advantages and disadvantages (Curram,94; Lim,97). The choice of a particular method depends, however, on factors like the kind of problem to be solved, the resources available, etc. An important factor in a good number of problems is the comprehensibility of the resulting classifier, that is, the possibility of understanding the resulting model and extracting useful knowledge to understand the modeled system. Approaches like neural networks and many of the statistical methods have very little comprehensibility (Weiss,91).

Another method, Fuzzy logic has been applied successfully (Fidelis,00; Gonzalez,98; Ishibuchi,00, 97 and 95) to extract comprehensible classifier knowledge from data in the form of linguistic rules (in this context, the linguistic is synonym of fuzzy). The fuzzy method has the ability to represent imprecise knowledge and the capability of dealing with noisy data.

Moreover, there have been several works that have attempted to produce classifier rules (fuzzy and non-fuzzy) using evolutionary techniques (Bojarczuk,99; Ishibuchi,95; Liu,00). One of the main problems encountered in this approach, is the representation of the condition part of a rule in the chromosome. Since the condition part can be a very complex logical expression, there is not a natural way to represent it as linear string. However, there are two main approaches that have been studied (De Jong,91):

- *Linear representation of the condition part*

  In some approaches (De Jong,91; Fidelis,00; Gonzalez,1998; Ishibuchi 00 and 95; Liu,00), the condition part was restricted to be a conjunction of one or more logical terms (tests). This makes the representation of the condition as a linear string. But, in general, a single rule is not sufficient enough to characterize a class; rather a set of rules is necessary. In other approaches (Giordana, 93), condition structures were predefined and only some parameters of rules were evolved. Figure 1(a) shows an example of such cases.

- *Tree representation of the condition part*

  In this approach, it was possible to represent arbitrarily complex conditions using Genetic Programming, with a substantial increase in the implementation complexity (Bojarczuk,99; Folino,99; Freitas,97; Tunstel,96). Figure 1(b) gives an example of tree representation.

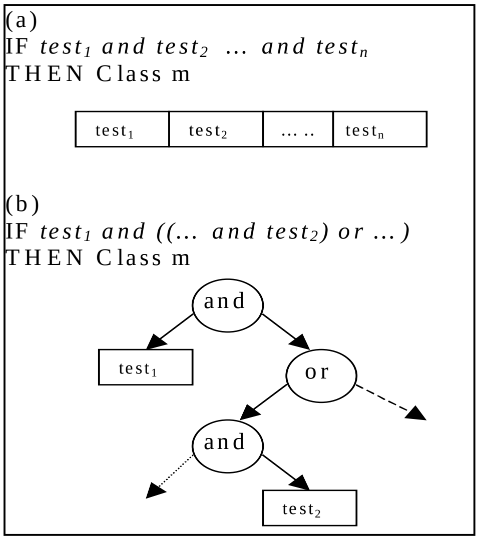

(a)
IF test1 and test2 ... and testn
THEN Class m
test1
test2
… ..
testn
(b)
IF test1 and ((... and test2) or ...)
THEN Class m
and
test1
or
and
test2

Figure 1: Conventional approaches to represent condition part of a rule. (a) a linear representation (b) a tree representation.

The purpose of the work presented in this paper, is to explore a new representation for linguistic classifier rules, which tries to combine the linear and tree methods, exploiting the advantages of both. In this approach, we evolve arbitrarily complex rules using a novel representation of tree structures in order to apply a genetic search.

The subsequent sections are organized as follows. Section 2 briefly describes the approach to perform classification tasks using fuzzy IF-THEN rules. Section 3 presents the proposed fuzzy rule representation scheme, Section 4 describes experiments and the analysis of results, and Section 5 draws some conclusions.

2 CLASSIFICATION USING LINGUISTIC RULES

In general, a linguistic classifier rule has the following form:

IF $x_1 \in S_1$ op_1 $x_2 \in S_2$ ... op_{n-1} $x_n \in S_n$ THEN *Class m*

where,

$\underline{x_i} \in [0.0,1.0]$, is an attribute or linguistic variable

$S_i \in \{S,MS,M,ML,L\}$, is a fuzzy set

$op_i \in \{AND, OR\}$, is a Fuzzy-Boolean operator

In this work, the attribute values are normalized in the interval [0.0,1.0] and fuzzy sets are defined by the membership functions shown in the Figure 2.

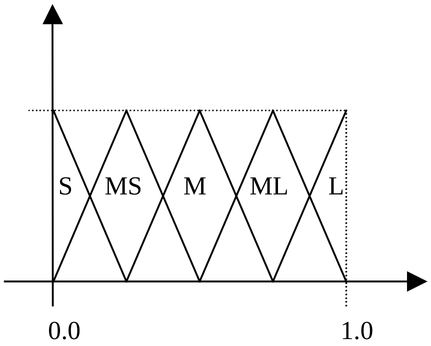


Figure 2: Fuzzy sets and membership functions

In our current experiments, we used 5 linguistic values, such as S (small), MS (medium small), M (medium), ML (medium large) and L (large). However, the method can be easily extended to any number of fuzzy values.

A classifier model can be represented by a set of *m* rules, where *m* is the number of different classes, that is, each class is represented by one, and only one, rule. For example,

R_1: IF $Condition_1$ THEN Class C_1

: : : :

R_m: IF $Condition_m$ THEN Class C_m

In order to classify an unclassified element $(x_1, \ldots , x_n)$, which is represented by a vector of attributes, the condition part of each rule is evaluated using the membership functions and the fuzzy-set operators[1]. Then, the rule with the highest value in the condition is selected, and the element is classified according to the consequent part of that rule:

$$Class(x_1,...,x_n) = \max_{c\in\{1,..,m\}} \{Condition_c(x_1,...,x_n)\}$$

where, $Condition_m(x_1, \ldots , x_n)$ represents the value of the $Condition_C$ evaluated for the element $(x_1, \ldots , x_n)$, which is a real value between 0.0 and 1.0.

3 PROPOSED APPROACH

In general, the condition part of a rule corresponds to a logic expression, which can be represented by an expression tree; a linear chromosome with variable length represents this expression tree.

A standard genetic algorithm with special operators is applied to evolve the rules. A GA run evolves a rule, so multiple runs are needed to cover all classes in the training set. The elements in the training set that belong to

---

[1] The union (OR) operator is calculated by the function *max(,)* and the intersection (AND) by the function *min(,)*

the class of the respective run are considered positive examples and the elements that belong to other classes are considered negative examples.

3.1 LINEAR REPRESENTATION OF LINGUISTIC RULES

Since there are different GA runs for each class, we do not have to represent the action part of the rule in the chromosome; it only represents the condition part.

Formally, a condition is generated by the following grammar:

(1) `<condition>` ::= `<condition> <operator> <condition>`
| `<atomic_condition>`
(2) `<atomic_cond>` ::= `<variable> <rel op> <set>`
(3) `<operator>` ::= `AND <prec> | OR <prec>`
(4) `<variable>` ::= $X_1 | ... | x_n$
(5) `<rel op>` ::= $\in$ | $\notin$
(6) `<set>` ::= `S | MS | M | ML | L`
(6) `<prec>` ::= `1 | 2 |...| 8`

The tree structure of an expression is generally expressed using braces that indicate the order of evaluation of the operators. When braces are not used, the default precedence of the operators determines the order of evaluation.

In our approach, we introduced precedence values for each operator in the representation itself (represented by `<prec>` in the grammar). This precedence value indicates the order of evaluation; an operator with a higher precedence value is evaluated first. Therefore, it is not necessary to have braces or a tree representation to express the evaluation order, so the expression can be represented by a linear string.

For example, the condition

$$X_2 \in MS\ AND_2\ X_1 \notin S\ OR_1\ X_3 \in ML\ AND_3\ X_2 \in L$$

represents the condition expression, as shown in Figure 3:

$$(X_2 \in MS\ AND\ X_1 \notin S)\ OR\ (X_3 \in ML\ AND\ X_2 \in L)$$

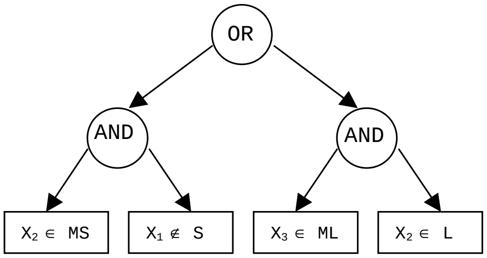


Figure 3: Tree representation of a condition expression

The precedence value of the operator AND_2 indicates that this operation has to be performed before the operation OR_1. When two consecutive operators have the same precedence value, the left one is evaluated first.

This scheme allows the representation of arbitrary complex conditions; the number of different precedence values determines the maximum depth of an expression tree.

Applying the grammar rule (1) multiple times, we get a condition with the following structure:

$$<ac_1> <op_1> \ldots <ac_n> <op_n> <ac_{n+1}>$$

where

$<ac_i>$: Atomic Condition
$<op_i>$: Fuzzy Operator

This condition expression is represented by a chromosome with the structure shown in the Figure 4.

$Gene_1$					...	$Gene_n$					$Gene_{n+1}$			
ac_1			op_1		...	ac_n			op_n		ac_{n+1}			**
var_1	ro_1	s_1	o_1	$prec_1$	...	var_n	ro_n	s_n	o_n	$prec_n$	var_{n+1}	ro_{n+1}	s_{n+1}	**

Figure 4: Chromosome representation of the condition.

An Atomic Condition and a Fuzzy Operator compose a gene. However, there is an exception in the last gene, which is composed of an Atomic Condition, and the last part (Fuzzy Operator) is ignored.

In our implementation, each gene is represented using 16 bits in the following way:

- Atomic Condition part:
  - 8 bits to represent the variable (var_i)
  - 1 bit to represent the relational operator (ro_i)
  - 3 bits to represent the set (s_i)
- Operator part:
  - 1 bit to distinguish between AND and OR (o_i)
  - 3 bits to represent the precedence ($prec_i$)

An important characteristic of this representation is that, in order to express the genotype, it is not necessary to build the expression tree. Instead, the classical parsing algorithm, *operator precedence parser* (Aho,86), can be used. This technique allows the evaluation of an expression in a very efficient way. The chromosome only has to be traversed once, that is, the time complexity of the evaluation is $O(n)$, where n is the condition expression length.

The evaluation algorithm based on the operator precedence parser can efficiently be implemented (using array and stack operations instead of pointer operations).

This fact along with the compact chromosomal representation makes this approach computationally inexpensive.

3.2 FITNESS EVALUATION

The fitness of each chromosome (rule) is evaluated with respect to a set of attribute vectors (training set) to which a class has been previously assigned. In each run of the genetic algorithm, a rule with different class C_i is evolved. Accordingly, vectors in the training set with class part equal to C_i are considered positive examples, and the elements with class part different from C_i are considered negative examples.

In our approach, the first step is to evaluate the condition part of the rule for a given vector. If the result is greater than or equal to 0.5, then the condition is true, otherwise it is false. Next, the class of the vector is compared to the class C_i of the actual run, and four different outcomes are possible, shown in Table 1.

Table 1: Types of the classifications results

Condition	Class	Type
TRUE	Equal	True Positive (TP)
TRUE	Different	False Positive (FP)
FALSE	Equal	False Negative (FN)
FALSE	Different	True Negative (TN)

The fitness of the condition is evaluated taking into account three objectives: maximize the sensitivity, maximize the specificity, and minimize the length of the chromosome. The length of the chromosome is penalized, because we want to evolve simple rules. This is an important factor that contributes to the comprehensibility. The formulas used are as follow:

$$sensitivity = \frac{TP}{TP + FN}$$

$$specfcity = \frac{TN}{TN + FP}, \text{ and}$$

$$fitness = w_1 \cdot sensitivity + w_2 \cdot specfcity + w_3 (1 - \frac{length}{MaxLength})$$

where *MaxLength* is the maximum allowable genes in a chromosome, and *length* is the actual number of genes in the chromosome.

This is a multi-objective problem, and there are different ways to deal with this kind of problem (Fonseca, 97). We chose to use a weighted sum approach, however, further experimentation with other multi-objective optimization approaches will be necessary. The w_i terms in the fitness definition represent the weight values.

3.3 GENETIC OPERATORS

The following genetic operators are used:

- **Restricted Crossover**: A crossover point is chosen between 1 and the minimum of the lengths of the two selected chromosomes. The child with minimal length is chosen (Figure 5.a).
- **Mutation**: A randomly chosen bit is changed as used in simple GA's.
- **Gene Elimination**: A gene is chosen randomly and eliminated. The length of the new chromosome is 16 bits shorter than the parent chromosome (Figure 5.b).
- **Gene Addition**: A random gene is generated and added at the end of the chromosome. The length of the new chromosome is 16 bits longer than the parent chromosome (Figure 5.c).

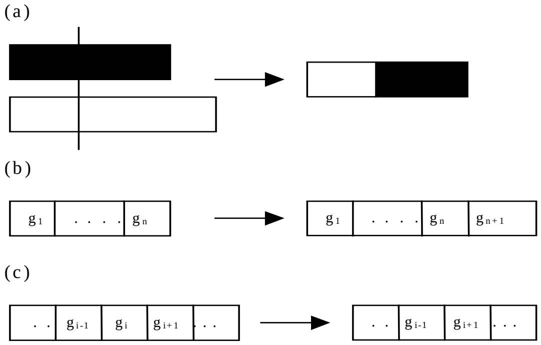


Figure 5: Genetic operators. (a) restricted crossover (b) gene addition (c) gene elimination.

However, only one operator is applied each time. The operator to be applied is chosen using a uniformly generated random number and the probability assigned to each operator.

4 EXPERIMENTATION

In order to evaluate the performance of the proposed approach to extract comprehensible linguistic rules from the training data, tests were conducted using publicly available data sets (University of California, Irvine, Repository of Machine Learning Databases (Blake,98)). These data sets are referenced frequently in the classification and machine learning literature, and it is a well-known standard for testing.

The data sets used are described in Table 2. The sample size s, the number of classes, and the type of attributes are shown in Table 2.

Table 2: Test Data sets used for experiments

Data Set	Sample size	No. of classes	Attributes	
			Numerical	Categorical
IRIS	150	3	4	0
VOTE	435	2	0	16
WINE	178	3	13	0

4.1 EXPERIMENTAL SETTING

The data sets with numerical attributes were normalized to have all values in a fixed range [0.0,1.0]. The attributes of the VOTE data set have 3 possible values 'YES', 'NO' and '?'[2]; these values were codified as 1.0, 0.0 and 0.5, respectively, to deal with categorical data.

A ten-fold testing strategy was employed (Lim, 97), that is, the data set was partitioned into ten randomly chosen subsets, and each subset was used as a testing set for the classifier trained with the remaining subsets. The score of the classifier (*correctly classified samples / sample size*) was calculated as the average score of 10 tests. This process was repeated 5 times for each data set and the average score was taken.

A number of GA parameters were tested, and the reported results used tournament selection, with a tournament size of 4, along with elitism -- the best individual of each generation is copied to the next generation.

GA parameter values:

Population:	200
Generations:	200
Mutation Rate:	0.05
Crossover Rate:	0.35
Gene Addition Rate:	0.35
Gene Elimination Rate:	0.25
Maximum Length:	50 genes

Each GA run was initialized with a random population of rules with five genes. The weights used in the fitness function were w_1=0.45, w_2=0.45 and w_3=0.1, to give more importance to sensitivity and specificity terms. In our empirical study, these values produced good results in different experiments; however, more experimentation will be necessary to define criteria for tuning parameter values.

4.2 RESULTS AND ANALYSIS

The average score and the variance in data sets are reported in Table 3. In particular, the value 94%+/-0.3 in the first row illustrates that the experimentations of the IRIS data set produce an average score of 94.5% with a variance of 0.3%. Although quantitative comparisons with other methods are useful, and desirable, our results compare well to those reported in the literature (IRIS (Folino,99; Gonzalez,1998; Ishibuchi,95; Liu,00), VOTE (Folino,99; Lim,97), WINE (Ishibuchi,00)).

Table 3: Results of average prediction accuracy

Data Set	Score
IRIS	94.5% +/- 0.3
VOTE	94.7% +/- 0.1
WINE	93.9% +/- 0.7

The most important objective of our fuzzy rule classification was to obtain comprehensible rules. The proposed approach was able to evolve simple rules, and the following set of rules were evolved for IRIS data set in a typical run:

R1: if $X_3 \in S$ OR_6 $X_2 \in S$ THEN Class 1

R2: if $X_3 \in M$ THEN Class 2

R3: if $X_3 \in ML$ OR $(X_2 \in ML$ AND $X_0 \in M)$ OR $X_3 \in L$ THEN Class 3

There are two key points to be noted in this set of rules:

- The simplicity of the rules: The system was able to evolve short rules.
- The feature selection: The system was able to select attributes that contribute to distinguishing a class, discarding other attributes.

These improved results were accomplished due to the evolutionary pressure towards shorter rules. Figure 6 shows the evolution of the fitness and the average rule size of the population in a particular run of the GA using the IRIS data set. It shows how the size of the rule converges to an optimal value, while the average fitness of the population increases.

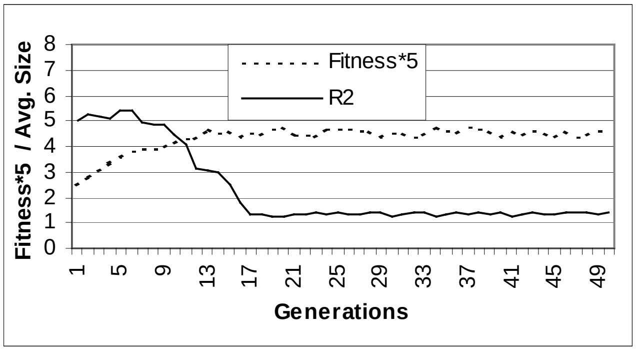


Figure 6: Fitness and size of the rule 2 evolution for the IRIS data set (the fitness is multiplied by 5)

---

[2] The character '?' means a neutral vote, neither YES or NO.

The optimal size of a rule depends on the complexity of the class that it intends to model. Figure 7 shows the change in the average size of three rules for the IRIS data set. The convergence value of each rule is different and it can be interpreted as a measure of complexity of the class that it represents.

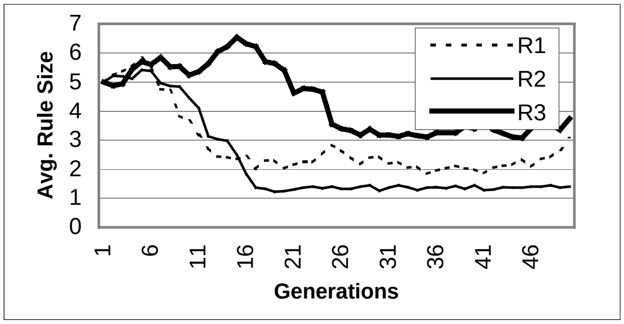


Figure 7:Evolution of the rule size for the IRIS data set.

Figure 8 presents the rule size behavior for the WINE data set. The convergence values are greater than those of the IRIS data set. This indicates that the WINE data set is more complex than the IRIS data set from the classification point of view.

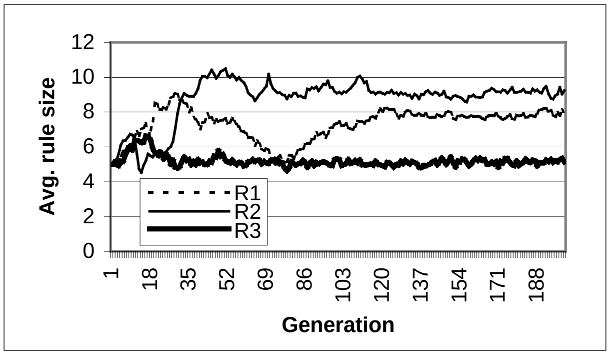


Figure 8: Evolution of the rule size for the WINE data set.

In some cases, there is a tradeoff between simplicity and accuracy. Table 4 shows the scores for the previously shown set of evolved rules for the IRIS data set. Here, the rule R2 has 4 false positives and 2 false negatives. In another run with the same training set (IRIS), a different rule was evolved for the class 2:

R2′:IF ($X_2 \in$ M OR $X_1 \notin$ S) AND $X_3 \in$ M

THEN Class 2

Table 4: Fitness of evolved rules for the IRIS data set. Here ***Sens, Spec*** and ***Fitn*** represent sensitivity, specificity and fitness, respectively.

Rule	**TP**	**TN**	**FP**	**FN**	**Sens**	**Spec**	**Fitn**
R1	42	93	0	0	100%	100%	0.996
R2	44	85	4	2	96%	96%	0.958
R3	47	80	8	0	100%	91%	0.951

This rule (R2′) has a more restrictive condition than previous R2. In effect, the number of false positives is reduced to 3, but the fitness remains similar (0.959) because of the increase of the condition length (chromosome).

The simplicity of the rules depends on the data set characteristics. For instance, the rules evolved for the WINE data set are more complex. The following is a typical rule evolved for this data set:

IF

($X_3 \in$ MS OR $X_{10} \in$ M) AND ($X_0 \notin$ MS OR $X_{10} \in$ S) AND $X_5 \notin$ MS AND $X_9 \notin$ S AND $X_6 \notin$ MS AND $X_4 \notin$ S

THEN Class 1

In the case of the VOTE data set, there are only two classes. The complexity of these two rules is expected to be the same, and the experiments confirmed that as is shown in figure 9.

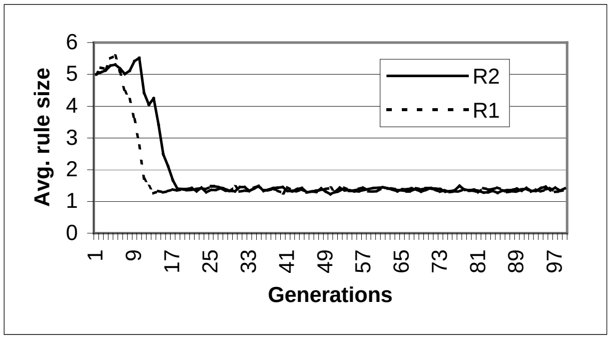


Figure 9: Average rule size for the VOTE data set.

5 CONCLUSIONS

Our experiments showed that the proposed representation works well in a wide variety of classification problems. Despite the fact that only five values for the linguistic variables were used, the accuracy of the evolved classifier rules was very good and comparable to those reported in the literature. The accuracy can be further improved by increasing the number of linguistic values and applying genetic tuning methods to the membership functions (Herrera, 98).

The main goal of this work was to evolve comprehensible rules, which could be accomplished by producing shorter rules, and performing automatic feature selection according to the complexity of data.

The main contribution of the present work is the design of a representation scheme. It allows an efficient and compact representation of complex conditions, using a linear chromosome.

However, more experiments need to be performed with bigger data sets and using other genetic operators. It is also important to perform quantitative comparison against other rule evolution methods, which is a part of our future work.

Acknowledgements

This work was funded by the Defense Advanced Research Projects Agency (no. F30602-00-2-0514) and Office of Navel Research (no. N000149910721).

References

A.V. Aho, R. Sethi, and J.D. Ullman (1986). *Compilers: Principles, Techniques, and Tools.* Addison-Wesley.

C.L. Blake, and Merz, C.J. (1998). UCI Repository of machine learning databases Irvine, CA: University of California, Department of Information and Computer Science. [http://www.ics.uci.edu/~mlearn/MLRepository.html].

C.E. Bojarczuk, H.S. Lopes and A.A. Freitas (1999). Discovering comprehensible classification rules using genetic programming: a case study in a medical domain. *Proc. Genetic and Evolutionary Computation Conference GECCO99*, Morgan Kaufmann, 1999, pp. 953-958.

S. Curram and J. Mingers (1994). Neural Networks, Decision Tree Induction and Discriminant Analysis: An empirical comparison. *J. of the Operational Research Society*, **45**(4):440-450.

K. De Jong and W. Spears (1991). Learning Concept Classification Rules Using Genetic Algorithms. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 651-656.

M.V. Fidelis, H.S. Lopes and A.A. Freitas (2000). Discovering comprehensible classification rules with a genetic algorithm. *Proc. Congress on Evolutionary Computation (CEC)*, pp. 805-810.

G. Folino, C. Pizzuti and G. Spezzano (1999). A Cellular Genetic Programming Approach to Classication. *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, Morgan Kaufmann, pp. 1015-1020.

C.M. Fonseca and P.J. Fleming (1997). Multiobjective Optimization. In *Handbook of Evolutionary Computation*, release 97/1, IOP Publishing Ltd. and Oxford University Press.

A.A. Freitas (1997). A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalised Rule Induction. *Proc. of 2nd Annual Genetic Programming Conference GP'97*, pp 96-101.

A. Giordana and L. Saitta (1993). Regal: an integrated system for learning relations using genetic algorithms. *Proceedings of the Second International Workshop on Multistrategy Learning*. R.S. Michalski et G. Tecuci (eds), pp. 234-249.

A. Gonzalez and R. Prez (1998). Completeness and consistency conditions for learning fuzzy rules. *Fuzzy Sets and Systems*, 96: 37-51.

J. Han and M. Kamber (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

F. Herrera, M. Lozano and J.L. Verdegay (1998). A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems*, 100:143-158.

R. Holte (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63-91.

H. Ishibuchi and T. Nakashima (2000). Linguistic Rule Extraction by Genetics-Based Machine Learning. *Proceedings of the Genetic and Evolutionary Computation Conference GECCO'00*, 195-202. Morgan Kaufmann.

H. Ishibuchi and T. Murata (1997). A genetic-algorithm-based fuzzy partition method for pattern classification problems. In F. Herrera and J.L. Verdegay (eds.), *Genetic algorithms and soft computing*, Physica-Verlag, pp. 555-578.

H. Ishibuchi, K. Nozaki, N. Yamamoto and H. Tanaka (1995). Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, **3**(3):260-270.

T. Lim and W. Loh (1997). *A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms*. Technical Report, Department of Statistics, University of Wisconsin-Madison, No. 979.

J. Liu and J. Kwok (2000). An extended genetic rule induction algorithm. *Proceedings of the Congress on Evolutionary Computation (CEC)*, pp.458-463.

R.S. Michalski, I. Bratko and M. Kubat (1998). *Machine learning and data mining: methods and applications*. J. Wiley & Sons, U.K.

E. Tunstel and M. Jamshidi (1996). On Genetic Programming of Fuzzy Rule-Based Systems for Intelligent Control. *Intl. Journal of Intelligent Automation and Soft Computing*, **2**(3):271-284.

S. M. Weiss and C. A. Kulikowski (1991). *Computer Systems that Learn. Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann Publishers, Inc.

Adaptive Mutation for Semi-Separable Problems

Mourad Elhadef
Department of Mathematics and Computer Science
University of Sherbrooke, Québec, CANADA.
Elhadef@dmi.usherb.ca

David A. Coley
School of Physics, University of Exeter,
Exeter, EX4 4QL, UK.
D.A.Coley@ex.ac.uk

Abstract

In this paper we introduce a new mutation heuristic in an attempt to better match genetic algorithms and the geography of search spaces. This is achieved by varying the mutation rate across the genotype to more rapidly search those areas that are currently believed to be having the greatest detrimental impact on the phenotype fitness. The new adaptive mutation operator is shown to be efficient in two applications: fault diagnosis in distributed and multiprocessor systems and the classical traveling salesman problem. We believe that the proposed, adaptive, mutation operator is the first step in realizing a new class of adaptive genetic operators for use with a distinct, but common, subset of real world applications.

1 INTRODUCTION

In the general case, the fitness landscape of a difficult multidimensional optimization problem is a complex multidimensional surface, exhibiting features over a wide-range of scales. For many problems this makes locally based search impractical and suggests the use of non-local methods; with one possibility being an evolutionary algorithm. Such algorithms are general, and have been applied to numerous problems such as numerical function optimization, combinatorial optimization, image processing, fuzzy logic, engineering processes, biology, artificial life, and machine learning (PPS, 1992), (PGA, 1993), (ICE, 1994), (PPS, 1994) (to name only a few). But, and moving away from the general case, many real-world fitness landscapes contain a large amount of (unknown) underlying structure. The use of a non-local method is still required for successful navigation through such a space, but ideally this should be with true regard to the geography of the landscape.

This paper describes a method of using some of this *geographic* information to improve the performance of genetic algorithms on certain problems. The class of problem the approach is applicable to are those that can naturally be expressed as:

$$fit_i = fit_{i1}(A_1) + fit_{i2}(A_2) + \ldots + fit_{in}(A_n) \qquad (1)$$

where i denotes an individual within the population, fit_i is the objective, or cost, function of the problem at hand, fit_{ij} a series functions which define fitness for subset A_j of unknowns P, i.e., $A_j \subset P$, and $n \leq$ the number of unknowns.

The classic example of such a problem is the Traveling Salesman Problem (TSP) (Lawer *et al.*, 1985), (Zbigniew, 1994), where the object to be minimized is the simple sum of the distances between the cities. Although to find the global optimum route is, for all but the simplest of arrangements, an extremely difficult problem, a simple heuristic such as "go to the next nearest city" works surprisingly well. The TSP is just one of a number of problems where we believe reasonable solutions can be found by separating the problem into a series of elemental fitnesses and treating these fragments differently within an evolutionary algorithm. The separation is only partial because the sub-sets in Eq. (1) share elements ($A_j \cap A_k \neq \emptyset; j \neq k$) and the fit_i are not completely independent. (For truly separable problems each unknown can be optimized sequentially and in isolation).

The remainder of this paper is organized as follows. In the next section we discuss the proposed adaptive mutation process and formalize it. In Sections 3 and 4 we present two applications of the adaptive mutation operator. Finally, we provide some concluding remarks and suggest future research directions.

2 ADAPTIVE MUTATION

In the early stages of a GA run, mutation has the power to provide novelty by vaulting individuals around the search space. In the later stages mutation is a background operator that ensures that the probability of finding the optimal solution is never zero and is useful for escaping from the local minima.

Mutation acts as a safety net to recover genetic material that may be lost through selection and crossover. The genetic material recovered may be “good” or “bad”. In this paper, we devise a new mutation process, which is expected to preferentially recover good genetic material. The new mutation process is henceforth referred to as *adaptive mutation* and the standard mutation operator as *random mutation*.

The key idea is to allocate an individual fitness to each gene in a chromosome, and thus, the whole fitness of the chromosome can be computed as function of its individual gene fitnesses. Consider a chromosome v and let $v[i]$ denotes the i^{th} gene. The fitness of gene $v[i]$ can be computed using a fitness function f. That is, $f(v[i])$ refers to the individual fitness of the i^{th} gene. The fitness of a given chromosome v can now be formalized as function of its individual gene fitnesses and is given by:

$$FT(v) = \Omega(f(v[1]), f(v[2]), \ldots, f(v[n]))$$

Rather than giving each gene an equal chance to be flipped, a *mutation threshold* is introduced. All genes with individual fitnesses not satisfying the threshold are mutated. It follows that the chance of a given gene undergoing mutation is now a function of its performance measure, i.e., its individual fitness. This will ensure a decreasing likelihood of mutating higher fitted bits, i.e., bits with individual fitness not satisfying the mutation threshold, as time goes by.

Although the method is not a general one, because of the need to have some rational way of temporarily assigning fitnesses to individual unknowns (or set of unknowns), it is our belief that many real world problems can be treated like this. In essence, it provides a natural way to apply varying mutation rates to different fragments of the genotype, which in turn, allows for a rapid search without restricting the size of the search space, or using varying length genotypes. The fragments of the genotype of most interest are those that impact most on the overall fitness.

Since mutation is now fitness-dependent, it follows that the mutation process should be performed before crossover. In the following sections, we present two applications of the new adaptive mutation operator, and demonstrate its efficacy.

3 APPLICATION I: FAULT DIAGNOSIS IN DISTRIBUTED AND MULTIPROCESSOR SYSTEMS

With advancing technology, distributed multiprocessor and multicomputer systems have arisen as powerful and economical tools for use in critical applications in military, commercial, and scientific computing. Along with this proliferation has come an increasing reliance on such systems, which unfortunately consist in most cases of inherently unreliable components. Therefore, it becomes essential to provide mechanisms for detecting and diagnosing faults at the system level. A crucial problem in this area, known as *system-level fault diagnosis*, is to identify the fault status of all processors in the system, i.e., to answer the question, which are faulty and which are fault-free? This problem has been extensively studied in the last two decades (Blough and Brown, 1999), (Dahbura, 1988), (Kreutzer and Hakimi, 1987), (Pelc, 1991).

The classical approach to fault diagnosis was originated by Preparata, Metze, and Chien in (Preparata *et al.*, 1967). They introduced a fault diagnosis model known as the PMC model, in which a set of intelligent, independent devices, called processors (or nodes), is assembled such that each node is tested by a subset of the other nodes. It is assumed that, at most, a bounded subset of these nodes is permanently faulty, and that it is possible that these faulty nodes can incorrectly claim that fault-free nodes are faulty or that faulty nodes are fault-free. A directed graph is used to model the testing assignment, where the set of vertices corresponds to the set of processing nodes and the set of edges (arcs) represents inter-node tests. From the collection of test results nodes are diagnosed as faulty or fault-free. Hakimi and Amin (Hakimi and Amin, 1975) gave necessary and sufficient conditions on the testing assignments of such systems so that, in spite of the presence of faulty nodes, each node can be correctly identified as faulty or fault-free based on any possible set of test results, assuming of course, that the number of faulty nodes does not exceed a given bound t. Such systems are said to be *t-diagnosable*. An example of a 4-diagnosable system is shown in Figure 1.

Although much is known about the nature of testing structures for diagnosable systems, the problem of efficiently identifying the set of faulty nodes of a system in which the fault situation is known to be diagnosable remains an outstanding research issue. Recently, a novel approach based on genetic algorithms have been developed to deal with this problem (Elhadef and Ayeb, 2000). The authors noted that a standard genetic algorithm requires many generations to identify the set of all faulty nodes, and hence, the diagnosis is not efficient since they aim on-line fault diagnosis. Adaptive mutation operator was developed to overcome

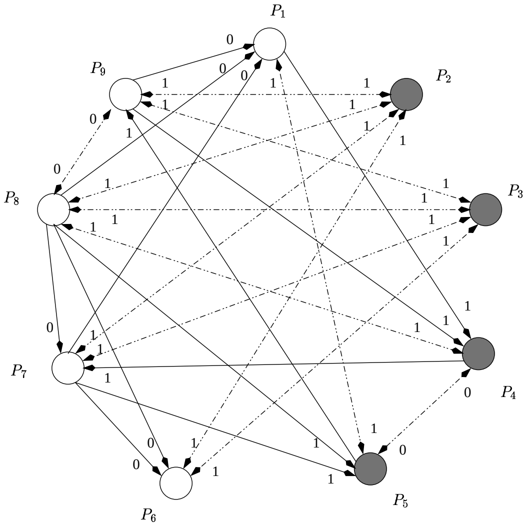


Figure 1: A testing assignment and a syndrome for a system which is 4-diagnosable. Faulty nodes are shaded. A dotted edge represents two nodes testing each other.

this problem.

3.1 MODEL DESCRIPTION

In the most general case, a distributed system is assumed to consist of a collection of n heterogeneous processing nodes, denoted by the set $U = \{P_1, \ldots, P_n\}$, interconnected via point-to-point communication links, broadcast busses, or an arbitrary combination of each. The system is modeled by two graphs: a *system graph* and a *testing graph*. A system graph is an undirected graph where every node represents a processor in the system, and an edge between two nodes represents a communication channel between the corresponding processors in the system. A testing graph is a directed graph which is a subgraph of the system graph. A directed edge from a node P_i to another node P_j in the testing graph means that P_i is a tester of P_j.

A node can be either *fault-free* or *faulty*, and its status does not change during diagnosis. Each node is assumed to be able to initiate a *test* of a neighboring node and to be able to respond to a test initiated by one of its neighbors. A test can be as elementary as an "Are you alive?" query. Each node $P_i \in U$ is assigned a particular subset of the remaining nodes to test, and it is assumed that no node test itself. The complete collection of tests in the system is represented by a testing graph $G(U, E)$. A test outcome w_{ij} is associated with each edge $(P_i, P_j) \in E$, where a weight of 0(1) is assigned to w_{ij} if P_i evaluates P_j to be fault-free (faulty). Since the faults considered are assumed to be permanent, the outcome w_{ij} of a test is reliable if and only if the tester node P_i is fault-free. Given a node $P_i \in U$, we denote by $\Gamma(P_i)$ the set of nodes tested by P_i and by $\Gamma^{-1}(P_i)$ the set of nodes testing P_i, and are given by:

$$\Gamma(P_i) = \{P_j : (P_i, P_j) \in E\}$$

$$\Gamma^{-1}(P_i) = \{P_j : (P_j, P_i) \in E\}$$

In addition, we associate to each node $P_i \in U$ the quantities $d_{in}(P_i) = |\Gamma^{-1}(P_i)|$ and $d_{out}(P_i) = |\Gamma(P_i)|$. The set of test outcomes of the system is called the *syndrome* and is denoted by S. Formally a syndrome is a mapping function $S : E \longrightarrow \{0, 1\}$, defined such as for all $(P_i, P_j) \in E, S(P_i, P_j) = w_{ij}$. We denote by $S(P_i)$ and $S^{-1}(P_i)$ the subsets of syndrome S corresponding, respectively, to tests carried out by node P_i and by $\Gamma^{-1}(P_i)$ and are defined by:

$$\begin{aligned} S(P_i) &= \{S(P_i, P_j) : P_j \in \Gamma(P_i)\} \\ S^{-1}(P_i) &= \{S(P_j, P_i) : P_j \in \Gamma^{-1}(P_i)\} \end{aligned}$$

A diagnosis algorithm is executed if some nodes become faulty at a given point of time. We consider only diagnosis of systems shown to be t-diagnosable, i.e., all faulty nodes can be unambiguously identified provided the number of faulty nodes is at most t.

3.2 FITNESS FUNCTION

A binary representation is used to represent the state (faulty or fault-free) of each node. For example, consider the system shown in Figure 1. The chromosome

$$\langle 100100010 \rangle$$

represents the set of faulty nodes $F = \{P_1, P_4, P_8\}$, i.e., those with a bit value at 1.

Consider a chromosome v, let F and S be its corresponding fault set and syndrome, respectively. We denote by $v[i]$ the i^{th} gene of chromosome v. Let F^* and S^* denote, respectively, the set of faulty nodes present at a given point of time and its corresponding syndrome. The performance of a single gene can be formalized as follows:

$$f(v[i]) = \frac{f_{out}(v[i]) + f_{in}(v[i])}{2}$$

where

$$f_{in}(v[i]) = \frac{|S^{-1}(P_i) \cap (S^*)^{-1}(P_i)|}{d_{in}(P_i)}$$

and

$$f_{out}(v[i]) = \begin{cases} 1 & \text{if } d_{out}(P_i) = 0, \\ \frac{|S(P_i) \cap S^*(P_i)|}{d_{out}(P_i)} & \text{otherwise.} \end{cases}$$

$f_{in}(v[i])$ computes the normalized number of tests performed on node P_i in syndrome S that are identical to their corresponding in syndrome S^*. Similarly, $f_{out}(v[i])$ calculates the normalized number of tests executed by P_i on its neighbors, i.e., $\Gamma(P_i)$. It follows that if node P_i has no neighbors, then $f_{out}(v[i]) = 1$; otherwise, $f_{out}(v[i])$ denotes the normalized number of test outcomes in syndrome S that match their corresponding in syndrome S^*. Note that the performance of the i^{th} bit includes both P_i's role: either as a tester, or as a testee. $f(v[i])$ can be seen as the correctness probability of the potential state of node P_i.

The objective function FT of the genetic algorithm used for diagnosis is given by:

$$FT(v) \quad = \quad \frac{\sum_{i=0}^{n-1} f(v[i])}{n}$$

Let $F(v)$ denotes the set of faulty nodes (those with a gene value at 1). It follows that if the chromosome v corresponds to the optimal solution, i.e., $F(v) = F^*$, then $FT(v) = 1$. The fitness function corresponds to the correctness probability of the potential solution $F(v)$. Since the set of faulty nodes F^* had to be uniquely identified it follows that the termination condition of the genetic algorithm is to achieve the fitness value 1. Therefore, the processes of selection, crossover, mutation, and evaluation are repeated until the particular chromosome corresponding to F^* is reached.

3.3 EXPERIMENTAL RESULTS

Simulation results showed that if every bit has an equal chance to undergo mutation (random mutation), a high number of generation is required in order to determine the optimal solution (see Table 1). Whereas, using adaptive mutation process, where genes are only flipped if and only if their individual performance measures are under certain threshold, the diagnosis algorithm is highly improved.

The mutation threshold was fixed to the value 0.5. Each bit has 50% of chance to undergo mutation since all bits are equiprobable. Furthermore, it is possible for some potential solutions that all genes have a performance measure superior to 0.5, in such a situation the bit with the lowest performance measure is flipped. This will ensure a decreasing likelihood of mutating higher fitted bits as time goes by. Note that a single-point crossover has been used.

4 APPLICATION II: THE TRAVELING SALESMAN PROBLEM

The now classic Traveling Salesman Problem (TSP) can also be formalized in order to be solved using adaptive mutation. This problem can be easily stated as follows: “suppose a salesman must visit clients in different cities, and then return home. What is the shortest tour through those cities, visiting each one once and only once?”. Evolutionary approaches to such problems have already been proposed in the literature (PPS, 1991), (Grefenstette, 1985), (Grefenstette, 1987), (Schaffer, 1989). Moreover, various genetic operators and different representations have been studied. In this section, our objective is first to show that adaptive mutation can be applied to TSP, and second to demonstrate the efficiency of this new method by comparing it with a standard random mutation.

Table 1: Number of generations needed to reach the optimal solution: (Random) every bit has an equal chance to undergo mutation and (Adaptive) the probability of mutation of the i^{th} bit is its performance measure $f(v[i])$. $pop\_size = 10$, number of runs = 1000.

n	Random	Adaptive	σ_{Random}	$\sigma_{Adaptive}$
8	57.95	2.74	50.29	2.21
16	48.89	2.39	25.30	1.13
32	157.03	3.71	64.69	1.85
50	203.72	4.47	62.58	2.01
100	586.13	6.45	144.98	3.35
200	$> 10^3$	11.91	$> 10^2$	6.02
500	$\gg 10^3$	22.13	$> 10^2$	13.16

4.1 PRELIMINARIES

We use *path representation* to encode the search space. Thus, a tour is represented by a list of n cities. Initial populations are generated randomly and a proportionate selection schema is used. Order crossover (OX) (Zbigniew, 1994) is used and elitism is incorporated in the genetic algorithm. We use inversion mutation in order to alter chromosomes. That is, if a gene has to undergo mutation another gene is chosen and both genes are swapped.

Let $\mathcal{M} = [c_{ij}]$ denotes a cost matrix, where c_{ij} represents the cost of traveling from city i to city j. We assume that the salesman can travel from any city to all other cities, and hence, the cost matrix is a square matrix $n \times n$. Before we present the fitness function we need first some kind of cost normalization. We use the following cost normalization function:

$$Cost(i,j) = \frac{\mathcal{M}[i,j]}{Max(\mathcal{M})}$$

where $Max(\mathcal{M})$ and $\mathcal{M}[i,j]$ denote, respectively, the maximum of cost matrix $\mathcal{M}$ and the cost of traveling from city i to city j. Using such normalization the maximum of the cost matrix is allocated the value 1 and the minimum of the cost matrix is allocated the lowest value over all. From now on, $\widetilde{\mathcal{M}}$ denotes the normalized cost matrix.

4.2 FITNESS FUNCTION

Consider a chromosome v, we denote by $v[i]$ the i^{th} gene of chromosome v. The fitness of a single gene can be formalized as follows:

$$f(v[i]) = \frac{Cost(v[i]) + Cost^{-1}(v[i])}{Cost(v)}$$

where $Cost(v[i])$ and $Cost^{-1}(v[i])$ denote, respectively, the cost of traveling from and to city $v[i]$, and $Cost(v)$ denotes the cost of the whole tour represented by chromosome v. $Cost(v), Cost(v[i])$, and $Cost^{-1}(v[i])$ are given by:

$$Cost(v) = \sum_{i=1}^{n} Cost(v[i])$$

$$Cost(v[i]) \quad = \quad \begin{cases} Cost(v[n], v[1]) & \text{if } i = n, \\ Cost(v[i], v[i+1]) & \text{otherwise.} \end{cases}$$

$$Cost^{-1}(v[i]) \quad = \quad \begin{cases} Cost(v[n], v[1]) & \text{if } i = 1, \\ Cost(v[i-1], v[i]) & \text{otherwise.} \end{cases}$$

Note that well fitted genes are those with lower fitness values. Now, we can define the fitness function FT to be maximized as follows:

$$\begin{aligned} FT(v) &= \frac{1}{Cost(v)} \\ &= \frac{2}{Cost(v) \sum_{i=1}^{n} f(v[i])} \end{aligned}$$

We also define a mutation threshold for each gene in a given chromosome as follows:

$$\delta(i) = \frac{Avg(\widetilde{\mathcal{M}}[i]) + Avg(\widetilde{\mathcal{M}}[i^-])}{Cost(v)}$$

where $Avg(\widetilde{\mathcal{M}}[i])$ denotes the average of raw cost corresponding to city i, and i^- corresponds to the city preceding city i in the tour represented by chromosome v.

4.3 SIMULATION RESULTS

Experimental results using adaptive mutation are compared with those generated using the standard random mutation operator. The pseudocode for random mutation is given in Figure 2 and that of adaptive mutation in Figure 3. The mutation rate is fixed to $p_m = 0.01$ and the population size to $pop\_size = 20$. Note that in adaptive mutation operator we flip genes that have individual fitnesses superior to their corresponding mutation thresholds, contrary to the standard mutation process. This is due to the meaning attributed to an individual fitness value. In fact, the individual fitness value $f(v[i])$ of the i^{th} gene represents the impact of that gene on the whole fitness of the given chromosome v. Hence, if a given gene has a high impact, it should be flipped since we deal with costs. To ensure that we are being fair with both mutation methods the number of mutations in the adaptive method is bounded by the expected number of mutated bits in the random method: $n \cdot p_m \cdot pop\_size$.

```
Procedure Random Mutation
Input  : A population of chromosomes $Pop$.
Output: A new population of chromosomes $Pop'$.
Begin
  for each chromosome $v \in Pop$ do
    for each gene $v[i]$ do
      $r \leftarrow$ random number from the range [0..1];
      if $(r < p_m)$ then
        Select another gene $v[j], j \neq i$;
        Swap(v[i],v[j]);
      endif
End
```

Figure 2: Pseudocode for random mutation.

```
Procedure Adaptive Mutation
Input  : A population of chromosomes $Pop$.
Output: A new population of chromosomes $Pop'$.
Begin
  $NumbMutations \leftarrow 0$;
  for each chromosome $v \in Pop$ do
    for each gene $v[i]$ do
      if $(NumbMutations < n \cdot p_m)$ then
        if $(f(v[i]) > \delta(v[i]))$ then
          Select another gene $v[j], j \neq i$ with the highest
            bit fitness value, i.e., $f(v[j])$, and which has
            not been mutated yet;
          Swap(v[i],v[j]);
          $NumbMutations++$;
        endif
      endif
End
```

Figure 3: Pseudocode for adaptive mutation.

We consider first the evolution of the population over generations. Next, we discuss the performance of the new mutation process.

4.4 POPULATION EVOLUTION

Figure 4 shows the evolution of the elite chromosome using a 100 × 100 cost matrix. We can conclude that adaptive mutation provides better results than random mutation. In fact, we can see that adaptive mutation finds better results faster than random mutation. On the other hand, Figure 5 shows the evolution of cost average over generations; again, we can see that adaptive mutation performs well. Under random mutation the cost average variation is decreasing slowly and it is not monotonic. Whereas, under adaptive mutation the cost average of generations is decreasing faster and the variation is somewhat monotonic. The number of mutations for both methods is plotted in Figure 6. It is interesting to note that although each method gives approximately the same number of mutation per generation (and hence we are being fair with both methods), adaptive mutation has a much smaller variation in number.

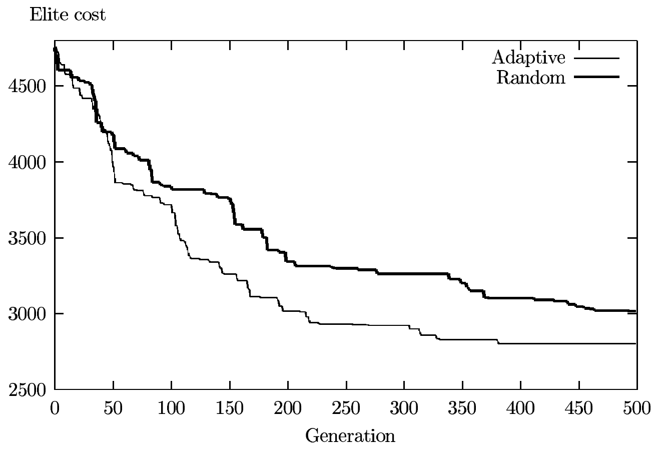


Figure 4: Evolution of the elite chromosome as a function of generation using a 100 × 100 cost matrix.

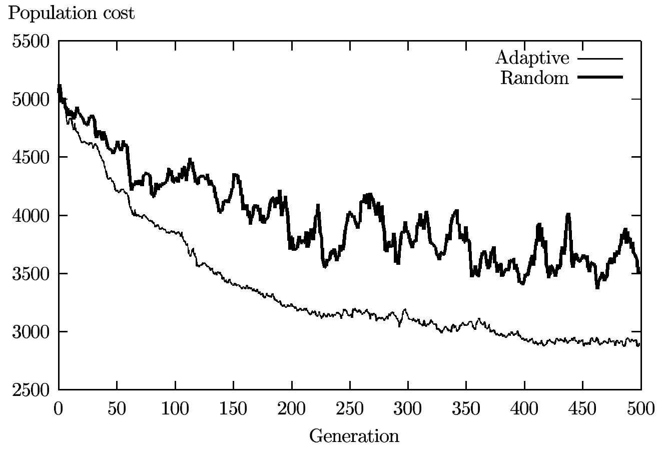


Figure 5: Evolution of the population average as a function of generation using a 100 × 100 cost matrix.

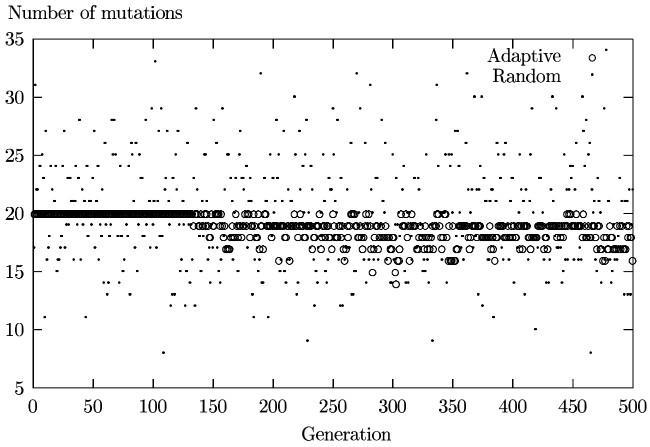


Figure 6: Number of mutations as a function of generation using a 100 × 100 cost matrix.

4.5 MUTATION EFFICIENCY

Extensive simulations were conducted to determine the performance of the new mutation process and specifically its impact on improving the efficiency of the genetic algorithm. The GA is performed 1000 times using different cost matrices. The number of generations was fixed to $nb\_gen = 200$. Each time both mutation operators start from the same randomly generated initial population. Experimental results are shown in Figure 7 using a 100 × 100 cost matrix. We can easily conclude that adaptive mutation provides better results than random mutation. In fact, all tours output using adaptive mutation imply lower cost than those output using random mutation. This result is clearer for large number of cities. Figure 8 shows a tour produced using adaptive mutation and a 20 × 20 cost matrix.

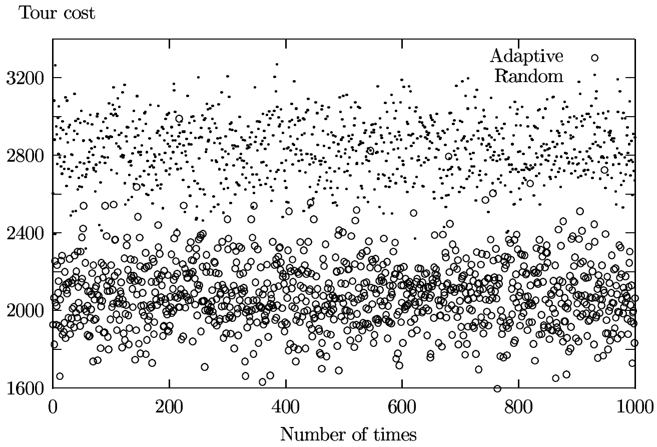


Figure 7: Tour costs produced after 200 generations using adaptive and random mutation and a 100 × 100 cost matrix.

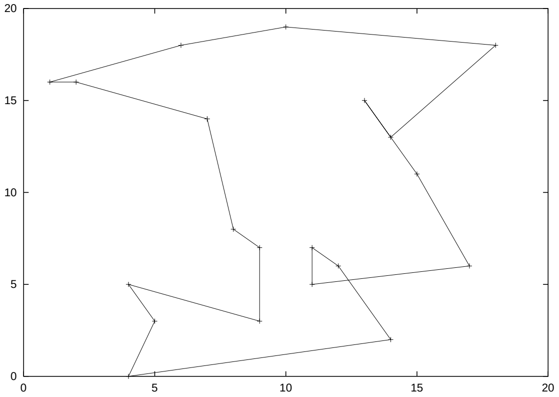


Figure 8: A tour produced after 500 generations using adaptive mutation and a 20 × 20 cost matrix.

5 CONCLUSION AND FUTURE RESEARCH

In this paper, we have presented a novel mutation heuristic. We have proposed that each gene in a chromosome should have an individual fitness measure that describes its impact on the whole fitness of the corresponding chromosome. Genes that do not satisfy some threshold are altered rapidly using mutation. The technique has been demonstrated successfully on two sample problems.

Future work will be directed at developing other new adaptive operators. For example, crossover is also typically considered as a random process, with crossover points determined randomly.

The technique is not a general one, but it can be applied to a variety of real world problems where there is a need to rapidly locate good solutions.

References

[Blough and Brown, 1999] D. M. Blough and H. W. Brown. The broadcast comparison model for on-line fault diagnosis in multiprocessor systems: Theory and implementation. *IEEE Trans. on Computers*, 48(5), 5 1999.

[Dahbura, 1988] A. Dahbura. System-level diagnosis: A perspective for the third decade. Technical report, AT&T Bell Laboratory Report, Concurrent Computations: Algorithms, Architecture, and Technology, S. Tewksbury, B. Dickinson, S. Schwartz, Eds. New York: Plenum, 1988.

[Elhadef and Ayeb, 2000] M. Elhadef and B. Ayeb. An evolutionary algorithm for identifying faults in $t-$diagnosable systems. In *Proc. of the 19th Symp. on Reliable Distributed Systems*, Nürnberg, Germany, 2000.

[Grefenstette, 1985] J.J. Grefenstette, editor. *Proc. 1st Int. Conf. Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[Grefenstette, 1987] J.J. Grefenstette, editor. *Proc. 2nd Int. Conf. Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[Hakimi and Amin, 1975] S. L. Hakimi and A. T. Amin. Characterization of the connection assignment of diagnosable systems. *IEEE Trans. on Computers*, C-23:1040–1042, 10 1975.

[ICE, 1994] *Proc. 1st IEEE Conf. on Evolutionary Computation*, Orlando, 1994.

[Kreutzer and Hakimi, 1987] S. Kreutzer and S. L. Hakimi. System-level fault diagnosis: A survey. *Microprocessing and Microprogramming*, 20:323–333, 5 1987.

[Lawer *et al.*, 1985] W. Lawer, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem*. John Wiley, Chichester, UK, 1985.

[Pelc, 1991] A. Pelc. Undirected graph models for system level fault diagnosis. *IEEE Trans. on Electron. Comput.*, 40(11):1271–1276, 11 1991.

[PGA, 1993] *Proc. 5th Int. Conf. Genetic Algorithms and Their Apllications*, University of Illinois at Urbana-Champaign, 1993.

[PPS, 1991] *Proc. 1st Conf. on Parallel Problem Solving from Nature*, Springer-Verlag, Lecture Notes in Computer Science, Vol. 496, 1991.

[PPS, 1992] *Proc. 2nd Conf. on Parallel Problem Solving from Nature*, Brussels, 1992.

[PPS, 1994] *Proc. 3rd Conf. on Parallel Problem Solving from Nature*. Springer-Verlag, 1994.

[Preparata *et al.*, 1967] F.P. Preparata, G. Metze, and R.T. Chien. On the connection assignment of diagnosable systems. *IEEE Trans. on Electron. Comput.*, 16(6), 12 1967.

[Schaffer, 1989] J. Schaffer, editor. *Proc. 3rd Int. Conf. Genetic Algorithms*, Morgan Kaufmann Publishers, Los Altos, CA, 1989.

[Zbigniew, 1994] M. Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, NY, 1994.

“Forging” Optimal Solutions to the Edge-Coloring Problem

Brandon P. Enochs
Math and Computer Science Dept.
University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189
brandon-enochs@utulsa.edu

Roger L. Wainwright
Math and Computer Science Dept.
University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189
rogerw@utulsa.edu

Abstract

The applications of a modified version of the simulated annealing algorithm for solving the edge-coloring problem, which consists of partitioning the edges of a graph into the minimum number of disjoint subsets such that no two edges in a given subset are adjacent, was investigated. With the exception of coloring bipartite graphs, finding a minimum edge coloring of a graph is NP-Complete. The traditional simulated annealing algorithm was modified to incorporate an additional perturbation (disruptive) operator. The details of the modifications to the simulated annealing algorithm, and the technique we chose to represent solutions for the edge-coloring problem are presented in the paper. We also describe two perturbation operators used by our modified technique, along with the heuristic function used to evaluate solutions. Our algorithm for solving the edge-coloring problem was tested on over sixty test graphs from the literature. In every case, our algorithm found the optimal edge coloring. We compared our algorithm to a grouping genetic algorithm technique for solving this problem published recently in the literature. Our results over a wide variety of test graphs were vastly superior to the grouping genetic algorithm technique.

1 INTRODUCTION

Generally, for NP-Hard partitioning problems like the vertex-coloring problem, heuristic optimization techniques, such as simulated annealing, are well suited for approximating solutions. The edge-coloring problem, which is another NP-Hard partitioning problem, is no exception. Solutions to the edge-coloring problem have numerous practical applications including timetable problems (Bondy & Murty, 1976), partitioning problems (Fiorini & Wilson, 1977), and certain types of scheduling problems. The edge-coloring problem consists of partitioning the edges of a graph *G* into the minimum number of disjoint subsets such that all edges in a given subset are not adjacent. The smallest number of subsets into which the edges of a graph can be partitioned is called the chromatic index of the graph *G*, denoted as $\chi'(G)$ (Fiorini & Wilson, 1977). The chromatic index of any graph is bounded by $\Delta_{max}(G) \leq \chi'(G) \leq \Delta_{max}(G) + \mu$, where $\Delta_{max}(G)$ is the maximum degree of the graph and μ is the maximum edge multiplicity of the graph (Vizing, 1964).

Khuri *et al.* (2000) provides the most recent and comprehensive work on various techniques for solving the edge-coloring problem. Khuri developed and tested three approaches for approximating solutions to instances of the edge-coloring problem. By far, Khuri’s most successful approach to approximating solutions to the edge-coloring problem came from his use of a grouping genetic algorithm. Grouping genetic algorithms, which are a subclass of genetic algorithms, focus on approximating solutions to problems with grouping properties, such as the edge-coloring problem. Using a grouping genetic algorithm, Khuri was able to find the optimal solution for nearly all of the graphs he tested. However, as the density, (which is the ratio of the number of edges to the number of vertices in a graph), increased, the performance of his approach faltered, producing approximations using as many as three additional colors beyond the optimal number.

In order to overcome the deficiencies experienced by Khuri’s algorithm, we developed a different approach to the edge-coloring problem. In our approach, solutions to the edge-coloring problem were “forged” using simulated annealing rather than “evolved” using genetic algorithms. We applied a slight modification to the traditional simulated annealing technique. This modification consisted of applying a secondary perturbation operator at key times during the execution of the algorithm. This modified technique was then applied to 63 simple graphs and 15 multigraphs from various sources in the literature such as Knuth’s Stanford GraphBase (1993) and the DIMACS ftp site (2000), and compared to the results obtained by Khuri (2000).

The rest of this paper is organized as follows: Section 2 presents the modifications we made to the traditional simulated annealing algorithm. Section 3 depicts the representation we used to encode a solution to the edge-coloring problem, and the definition of the heuristic function used to evaluate candidate solutions. The algorithms used by the perturbation, and the additional perturbation operators to improve solutions, as well as the details of how solutions are initially generated are also presented in Section 3. Section 4 presents the results of several runs of our modified simulated annealing algorithm on various the test graphs from the literature. Section 5 describes the conclusions of our research, and comparisons to results obtained Khuri's grouping genetic algorithm. Acknowledgements and references are given at the end of the paper.

2 SIMULATED ANNEALING AND MODIFICATIONS

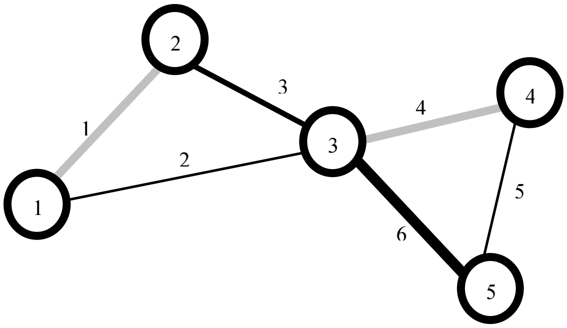
Traditionally, simulated annealing uses only one operation—perturbation. Perturbation, which is similar in functionality to mutation in genetic algorithms, attempts to improve a solution by making small modifications. The resultant solution created by perturbation is accepted or rejected based on several parameters given at runtime. For the edge-coloring problem, perturbation alone was not able to solve certain graphs in a reasonable amount of time; therefore, an additional operator was implemented to overcome this weakness. The new operator, which we call the "kick" operator, is similar in functionality to perturbation. The kick operator functions as follows: if a specified number of iterations has passed during which there has been no heuristic improvement, then some perturbation on the current solution is performed. This perturbation is intended to disrupt the current solution such that further iterations of the modified simulated annealing algorithm will lead to heuristic values exceeding those found before the kick was executed. Unlike the perturbation operation, the modifications made by the kick operator are accepted regardless of any benefits or losses that result. The modified simulated annealing algorithm is described as follows (it is assumed that the reader is familiar with simulated annealing):

1. Generate an initial solution and set the change counter to zero.
2. Given the parameters T, the initial temperature, α, the rate at which the temperature decreases, N, the initial number of iterations to perform before decreasing the temperature and increasing the number of iterations to be performed, β, the rate at which N increases, and C, the number of iterations allowed to occur in which the current solution does not improve before the kick operator is executed. Execute the following three steps N times.
    a. Execute the perturbation operator on the current solution. This will produce a new solution.
    b. Choose a new current solution by weighing the new solution against the current solution with the acceptance function. The acceptance function will choose one of the two solutions based on the current temperature, T, the heuristic value of the current solution, and the heuristic value of new solution. The acceptance function behaves as follows: If the new solution is better than the current solution, then the new solution is chosen and the change counter is reset to zero; however, if the new solution is worse than the current solution, a random real number is generated and compared against a value based on the current temperature and the heuristic difference between the current and new solutions. If the random real number is greater than or equal to the calculated value, then the new solution is chosen and the change counter is reset; otherwise, the current solution is chosen and the change counter is incremented by one (except for the change counter logic, this a traditional simulated annealing algorithm).
    c. After the acceptance function has chosen a new current solution, test the following condition: if the change counter equals C, then perform the kick operation and reset the change counter to zero.
3. Increase N to $N*\beta$, and decrease T to $T*\alpha$..
4. Perform the following test: if the total number of iterations performed equals the maximum number of iterations allowed, the maximum value of the heuristic is found, or a specified time limit is exceeded, then return the current solution, otherwise, repeat step 2a.

3 SOLUTION REPRESENTATION, HEURISTIC EVALUATION, AND OPERATOR ALGORITHMS

3.1 SOLUTION REPRESENTATION, HEURISTIC EVALUATION, AND OPERATOR ALGORITHMS

The solution representation is extremely important to the success of any algorithm. Our solution representation of the edge coloring of a graph is simply an array of integers representing the color assigned to each edge with the edges of a graph numbered *a priori*. This proved to be extremely successful. The first number represents the color of edge 1; the second number represents the color of edge 2; and so on. This representation lends itself to efficient color lookup, efficient heuristic evaluation, and efficient memory usage. Figure 1 shows an example graph and the solution representation for that graph.



SOLUTION

EDGE	1	2	3	4	5	6
COLOR	3	1	2	3	1	4

Figure 1: Solution Representation

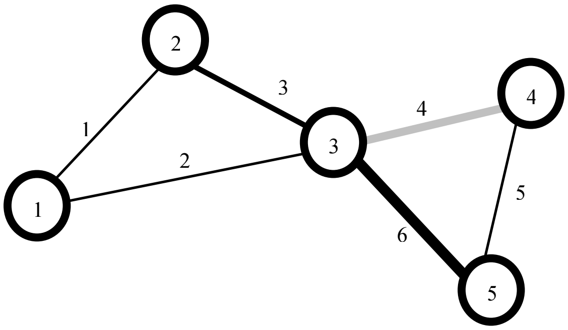
3.2 HEURISTIC FUNCTION

The heuristic function used for this problem is defined as $\Sigma\ [d(i) - invalid(i)]$ for $i = 1..n$, where $d(i)$ is the degree of the i^{th} node, $invalid(i)$ is the number of edges with the same color that are connected to the i^{th} node, and n is the number of nodes in the graph. It is important to note that $\Sigma\ d(i)$ for $i=1..n$ is a fixed constant for a given graph. Thus, when the value of the heuristic function is equal to $\Sigma\ d(i)$ for $i=1..n$, a solution is feasible. Therefore, the heuristic function measures a solution's distance to feasibility; consequently, higher heuristic values indicate solutions that are closer to feasibility. In addition, the number of usable colors is a fixed constant specified at run-time. Thus, annealing can stop when the heuristic value of the current solution is equal to $\Sigma\ d(i)$ for $i=1..n$, indicating that no two adjacent edges share the same color.

3.3 PERTURBATION FUNCTION

The pertubation function is a greedy operation. The perturbation operator randomly selects one node in which infeasible edges are incident. The operator then randomly selects one of those incident infeasible edges. Next, the operator attempts to assign a non-conflicting color to the selected edge. If a non-conflicting color cannot be found, then the selected edge is assigned a random color.

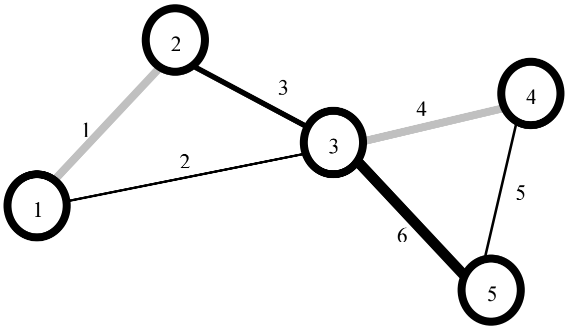
For example, consider Figure 2a. Assume that node 1 is randomly selected. Observe that edges 1 and 2, which are incident to node 1, have the same color, (color 1). Assume that edge 1 has been randomly selected. Then a free color (a color other than color 1 in this case) is randomly selected. Assume the free color selected was color 3, then color 3 is assigned to edge 1 (see figure 2b). If no free color exists, then edge 1 receives a random color that was not the color of any edge connected to node 3 to which edge 1 is adjacent (node 3 has the highest degree).



(a) BEFORE

SOLUTION

EDGE	1	2	3	4	5	6
COLOR	1 (invalid)	1	2	3	1	4



(b) AFTER

SOLUTION

EDGE	1	2	3	4	5	6
COLOR	3 (valid)	1	2	3	1	4

Figure 2: Perturbation Example with Node 1 Selected.

3.4 KICK FUNCTION

The kick function is a disruptive operation that reassigns a new color to a randomly selected edge with an invalid color. Empirical results support the claim that this operator is able to overcome many of the local hurdles that the heuristic function presents, producing the optimal solution on certain graphs in ¼ of the time required for traditional simulated annealing to produce the same results. On every graph we tested, traditional simulated annealing was able to produce an optimal solution. However, on many graphs, depending upon the seed used, traditional simulated annealing required as many as four times the number of iterations compared to our modified simulated annealing algorithm. As a kick operator example, consider again Figure 2a. In this case, edge 1 has been assigned an invalid color, and is, therefore, a candidate for the kick operation. Assume that the kick operator randomly chose edge 1. Edge 1 would be reassigned a random valid color, in this case, 3. If no color for edge 1 had existed, which was not in conflict with other colors assigned to adjacent edges, then edge 1 would have been assigned a random color.

3.5 INITIAL SOLUTION GENERATION

The initial solution is constructed using the following algorithm: first, find the node with the highest degree. Second, assign a different color to each edge connected to the node with the highest degree. Finally, the remaining edges are then assigned random colors. Note the range of valid colors for the remaining edges is reduced by the colors of edges connected to the node with the highest degree. For example, in Figure 1, node 3 has the highest degree; thus, edges 2, 3, 4, and 6 must always have different colors. As a result, this significantly reduces the search space for any given graph. Furthermore, the colors of the edges connected to the vertex with the highest degree do not change! This is an application of the reduction principle presented in (Corcoran & Wainwright, 1996), and proved to be significant to the success of our algorithm. Also the number of usable colors is a fixed constant specified by the user at runtime. Fixing the number of usable colors in this manner turned out to be a critical aspect of the success of our algorithm. Fixing the usable colors allows the user to specify a minimum quality for solutions. Thus, all the user has to do is specify the optimal number of colors to obtain an optimal solution

4 RESULTS

The performance of our modified simulated annealing algorithm and the grouping genetic algorithm were compared on 63 problem instances taken from various sources in the literature such as the DIMACS challenge ftp site (2000) and the Stanford GraphBase (1993). Graph density values ranged from 2 edges per node all the way up to 99.5 edges per node. In addition, graphs with a wide range of other properties, such as completeness, were tested. Table 1 depicts the results obtained for the 63 problem instances from our modified simulated annealing algorithm and the grouping genetic algorithm described by Khuri (2000). For the modified simulated annealing algorithm, the following parameters were used: $T = 0.0, \alpha = 0.0$, $\beta = 1.0$, $N = 1000$, and C=500...1250. For Khuri's grouping genetic algorithm, the following parameters were used: mutation rate = 0.2, crossover rate = 0.6, and population size = 20. The Problem Instance (left half) portion of Table 1 lists the datasets used and various properties of those datasets. The first 21 datasets are complete graphs named complete***x***.col, where ***x*** is the number of nodes in the graph. The remaining test graphs were taken from Donald Knuth's Stanford GraphBase (1993) and the DIMACS ftp site (2000). The column headed by Δ indicates the maximum degree for each graph, and, for many graphs, the minimum number of colors that can be used to color that graph.

The results shown in the right portion of Table 1 record the *# of iterations*, the *# of colors used,* and the *time required* (in seconds) to reach the optimal solution for our modified simulated annealing algorithm. We ran our modified simulated annealing algorithm on the test datasets using an Intel Pentium III 700Mhz processor running Microsoft Windows 2000 Professional and Sun's JDK version 1.3. Optimal solutions, which are indicated as a **bold** entry in the column (# of colors used), are solutions where the number of colors used were in the range of [Δ…Δ+μ], where μ is the maximum edge multiplicty of the graph.

Khuri *et al.* (2000) research represents the most recent work on this topic. In his paper he developed three techniques for solving the edge-coloring problem, the best of which was his GGA algorithm. The results from Khuri's GGA algorithm for the 63 test graphs are given in the last column under the heading *GGA results*. Similarly, entries in the column (GGA Results) that are depicted in **bold** represent optimal solutions. Our algorithm found the optimal solution in all 63 cases. Khuri's grouping genetic algorithm found the optimal solution in 21 of the 63 cases.

5 CONCLUSIONS

The results indicate that the edge-coloring problem lends itself very nicely to our modified simulated annealing algorithm. Impressively, our modified simulated annealing algorithm generated optimal solutions for every test data set. This includes all of the multigraphs from the Stanford GraphBase, the simple graphs from the DIMACS ftp site, and the generated complete graphs. The optimal coloring was even obtained for a complete graph with 200 nodes. For every dataset except those that formed a complete or regular graph with an odd number of nodes, the number of colors used in the final solution was equal to Δ, the optimal number. In the cases in which datasets formed a complete or regular graph with an odd number of nodes, the number of colors used increased to Δ+1; however, the chromatic index of a complete or regular graph, K_n, for which n is odd, is Δ+1, thus, all results in which Δ+1 colors were used were optimal as well (Rosen, 2000). The empirical results obtained using the modified simulated annealing algorithm support the claim that it is superior in performance when compared to recently developed algorithms for the edge-coloring problem. The use of only a fixed number of colors was of critical importance to the success of the modified simulated annealing technique in finding optimal solutions to the graphs tested. Limiting the number of usable colors allowed the modified simulated annealing technique to determine with absolute certainty whether or not the best solution, given the initial parameters, had been found. Also, by fixing the number of usable colors to the optimal number, the modified simulated annealing algorithm worked only towards finding the optimal solution while retaining the ability to produce feasible solutions that are not optimal. Also, the masking performed in assigning colors of edges reduced search space and lead to faster discoveries of optimal solutions. The incorporation of the kick operator, while unnecessary in producing optimal solutions, greatly reduced the average number of iterations required to find solutions to instances of the edge-coloring problem. Finally, the modified simulated annealing algorithm proved to be an excellent algorithm for solving the edge-coloring problem over a wide variety of graphs.

Acknowledgments

The authors wish to thank Sami Khuri and Tim Walters for providing us with additional results of their algorithm not published in their original paper, and for providing us with their software to run their GGA algorithm for solving the edge-coloring problem.

References

Bondy, J.A., & Murty, U.S.R. (1976). *Graph Theory with Applications*. New York: McMillan.

Corcoran, A.L. & Wainwright, R.L. (1996). Reducing Disruption of Superior Building Blocks in Genetic Algorithms. *Proceedings of the 1996 ACM/SIGAPP Symposium on Applied Computing* (pp. 269-276). Philadelphia, PA. ACM Press.

DIMACS: Center for Discrete Mathematics and Theoretical Computer Science (2000). ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color.

Fiorini, S. & Wilson, R.J. (1977). *Edge-Colourings of Graphs*. London: Pitman.

Holyer, I. (1980). The NP-completeness of edge coloring. *SIAM Journal of Computing*, 10, 718-720.

Khuri, S., Walters T., & Sugono, Y. (2000). A Grouping Genetic Algorithm For Coloring The Edges of Graphs. *Proceedings of the 2000 ACM Symposium on Applied Computing* (pp. 422-427). Villa Olmo, Como, Italy. ACM Press.

Knuth, D. (1993). *The Stanford GraphBase: A Platform for Combinatorial Computing*. New York: ACM Press.

Rosen, K. (2000). *Handbook of Discrete and Combinatorial Mathematics*. New York: CRC Press.

Vizing, V. (1964). On an estimate of the chromatic class of a p-graph (in Russian). *Diskret. Analiz.*, 3, 25-30.

Table 1: Edge Coloring Results

PROBLEM INSTANCE					RESULTS[1]			
File Name	# of nodes	# of edges	Density[2]	Δ	Iterations	# of colors used	Time (seconds)	GGA results
complete20.col	20	190	9.5	19	795	**19**	1	20
complete25.col	25	300	12	24	1321	**25**	2	26
complete30.col	30	435	14.5	29	1966	**29**	3	31
complete35.col	35	595	17	34	2697	**35**	4	37
complete40.col	40	780	19.5	39	4116	**39**	4	41
complete45.col	45	990	22	44	6106	**45**	5	47
complete50.col	50	1225	24.5	49	6870	**49**	6	52
complete55.col	55	1485	27	54	7233	**55**	7	58
complete60.col	60	1770	29.5	59	9973	**59**	11	63
complete65.col	65	2080	32	64	12817	**65**	17	70
complete70.col	70	2415	34.5	69	16789	**69**	23	75
complete75.col	75	2775	37	74	19892	**75**	35	78
complete80.col	80	3160	39.5	79	22407	**79**	43	83
complete85.col	85	3570	42	84	25563	**85**	52	88
complete90.col	90	4005	44.5	89	26638	**89**	77	93
complete95.col	95	4465	47	94	31124	**95**	92	98
complete100.col	100	4950	49.5	99	36267	**99**	127	104
complete105.col	105	5460	52	104	40065	**105**	169	109
complete110.col	110	5995	54.5	109	37493	**109**	180	114
complete115.col	115	6555	57	114	40967	**115**	233	119
complete200.col	200	19900	99.5	199	174191	**199**	3341	203
anna.col	138	986	7.14	142	224	**142**	2	**142**
david.col	87	812	9.33	164	113	**164**	1	**164**
homer.col	561	3258	5.80	198	590	**198**	5	**198**
huck.col	74	602	8.14	106	118	**106**	1	**106**
jean.col	80	508	6.35	72	125	**72**	1	**72**
mile250.col	128	774	3.02	32	470	**32**	1	**32**
miles500.col	128	2340	9.14	76	1985	**76**	4	77
miles1000.col	128	6432	50.25	172	6326	**172**	133	173
miles1500.col	128	10396	81.22	212	16563	**212**	171	213
queen5_5.col	25	320	12.8	32	377	**32**	1	33
queen6_6.col	36	580	16.11	38	867	**38**	1	39
queen7_7.col	49	952	19.43	48	1289	**48**	2	**48**
queen8_8.col	64	1456	22.75	54	2215	**54**	3	**54**
queen9_9.col	81	2112	26.07	64	2841	**64**	5	**64**
queen8_12.col	96	2736	28.5	64	5249	**64**	9	65
queen10_10.col	100	2940	29.4	70	4643	**70**	9	**70**

---

[1] Optimal solutions are indicated in bold.

[2] Density is the ratio of the # of edges to the # of vertices in a graph.

Table 1: Continued

PROBLEM INSTANCE					RESULTS[3]			
File Name	**# of nodes**	**# of edges**	**Density[4]**	**Δ**	**Iterations**	**# of colors used**	**Time (seconds)**	**GGA results**
queen11_11.col	121	3960	32.73	80	5540	**80**	15	**80**
queen12_12.col	144	2596	36.05	86	8136	**86**	33	**86**
queen13_13.col	169	6656	39.38	96	9897	**96**	56	97
queen14_14.col	196	8372	42.71	102	12869	**102**	96	**102**
queen15_15.col	225	10360	46.05	112	15047	**112**	143	113
queen16_16.col	256	12640	49.38	118	19731	**118**	243	119
myciel3.col	11	20	1.81	5	12	**5**	1	**5**
myciel4.col	23	71	3.09	11	25	**11**	1	**11**
myciel5.col	47	236	5.02	23	119	**23**	1	**23**
myciel6.col	95	755	7.95	47	326	**47**	2	**47**
myciel7.col	191	2360	12.36	95	820	**95**	3	**95**
games120.col	120	1276	10.63	26	1836	**26**	2	27
le450_15a.col	450	8168	18.15	99	4051	**99**	35	**99**
le450_15b.col	450	8169	18.15	94	4293	**94**	36	**94**
le450_15c.col	450	16680	37.06	139	12153	**139**	234	**139**
le450_15d.col	450	16750	37.22	138	12297	**138**	231	140
le450_25a.col	450	8260	18.35	128	3558	**128**	33	129
le450_25b.col	450	8263	18.36	111	4018	**111**	38	112
le450_25c.col	450	17343	38.54	179	9887	**179**	243	180
le450_25d.col	450	17425	38.72	157	12030	**157**	440	159
le450_5a.col	450	5714	12.7	42	4764	**42**	25	43
le450_5b.col	450	5734	12.74	42	4677	**42**	25	43
le450_5c.col	450	9803	21.8	66	9175	**66**	93	67
le450_5d.col	450	9757	21.68	68	8485	**68**	88	69
fpsol2.i.3.col	325	8688	26.73	346	3082	**346**	51	347

---

[3] Optimal solutions are indicated in bold.

[4] Density is the ratio of the # of edges to the # of vertices in a graph.

A Practical Schema Theorem for Genetic Algorithm Design and Tuning

David E. Goldberg and Kumara Sastry
Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
104 S. Mathews Ave, Urbana, IL 61801
`{deg,ksastry}@uiuc.edu`

Abstract

This paper develops the theory that can enable the design of genetic algorithms and choose the parameters such that the proportion of the best building blocks grow. A practical schema theorem has been used for this purpose and its ramification for the choice of selection operator and parameterization of the algorithm is explored. In particular stochastic universal selection, tournament selection, and truncation selection schemes are employed to verify the results. Results agree with the schema theorem and indicate that it must be obeyed in order to ascertain sustained growth of good building blocks. The analysis suggests that schema theorem alone is insufficient to guarantee the success of a selectorecombinative genetic algorithm.

1 Introduction

The importance of building blocks (BBs) and their role in the workings of GAs have long been recognized (Holland, 1975; Goldberg, 1989). Furthermore, the following six conditions for a GA success have been proposed (Goldberg, Deb, & Clark, 1992): (1) Know what GAs are processing - building blocks (BBs), (2) ensure an adequate initial supply of raw BBs, (3) ensure growth of superior BBs, (4) ensure the mixing of BBs, (5) ensure good decisions among competing BBs, and (6) solve problems with bounded BB difficulty. One of the important conditions is to make sure that the GA is well supplied with a sufficient supply of the BBs required to solve a given problem. It is also equally important that the proportion of the good ones in the population grow. The first task is addressed elsewhere in this proceedings (Goldberg, Sastry, & Latoza, 2001), and the second task, that is, the issue of guaranteeing the increase in market share of good BBs in a population is addressed in the current study.

The usual approach in achieving this is the schema theorem (Holland, 1975; De Jong, 1975). Therefore, the objective of this study is to utilize a practical schema theorem to explore the ramifications of the schema theorem for the choice of selection operator and parameterization of the algorithm. In this study we consider three selection schemes: stochastic universal selection (SUS) (Baker, 1987; Grefenstette & Baker, 1989), s-wise tournament selection (Goldberg, Korb, & Deb, 1989), and truncation selection (Muhlenbein & Schlierkamp-Voosen, 1993). SUS is a proportionate scheme and s-wise tournament selection and truncation selection are ordinal schemes. The performance of each of these selection schemes in both early as well as late in the GA run is analyzed based on the schema theorem.

We start by presenting a brief note on the schema theorem, both its original version, and a generalized version. A simplified version of the generalized schema theorem is then used for the BB growth design. Three different selection schemes are considered in the light of the BB growth design and are analyzed for parameter settings to ensure the growth of best BBs in the population.

2 Generalized Schema Theorem

There have been many studies on schema theorem, and a complete literature review on schema theorem is beyond the scope of this study. Instead, we present a brief overview of schema theorem and refer the reader elsewhere (Goldberg, 1989) for a detailed exposition of the same. Under proportionate selection, single-point crossover, and no mutation, the schema theorem may be written as follows:

$$\langle m(H,t+1)\rangle \geq m(H,t)\frac{f(H,t)}{\overline{f}(t)}\left[1-p_c\frac{\delta(H)}{\ell-1}\right], \quad (1)$$

where $\langle m\,(H,t+1)\rangle$ is the expected number of individuals that represent the schema H at generation $t+1$, $m(H,t)$ is the number of individuals that represent the schema H at generation t, $f(H,t)$ is the average fitness value of the individuals containing schema H at generation t, $\overline{f}(t)$ is the average fitness of the population at generation t, p_c is the crossover probability, $\delta(H)$ is the defining length defined as the distance between the outermost fixed positions of a schema, and ℓ is the string length.

Inspection of the schema theorem and an analysis of proportionate selection and single-point crossover (Goldberg, 1989), indicates that the the term $m\,(H,t)\,\frac{f(H,t)}{\overline{f}(t)}$ accounts for the selection, and the term $\left[1-p_c\frac{\delta(H)}{\ell-1}\right]$ accounts for crossover operation. It should be noted that the term representing the selection operator is exact and the inequality occurs due to the crossover operation. Some factors like crossover between identical individuals (self-crossover) are neglected. The schema theorem tells us that the proportion of schemata increases when they have above average fitness and relatively low crossover disruption.

However, the schema theorem as given by equation 1 is restricted to proportionate selection and one-point crossover. This concern can be eliminated by identifying the characteristic form of schema theorem and substituting appropriate terms for other selection schemes and genetic operators. Recognizing that a selection scheme might allocate the numbers of schemata in a different manner, and a genetic operator might yield a different survival probability when compared to proportionate selection and single point crossover, the following *generalized* schema theorem (Goldberg & Deb, 1991) can be written

$$\langle m(H,t+1)\rangle \geq m(H,t)\gamma(H,m_i,f_i,t), \quad (2)$$

where,

$$\gamma(H,m_i,f_i,t)=\phi(H,m_i,f_i,t)P_s(H,m_i,f_i,t), \quad (3)$$

and ϕ is the selection factor, and is a function of the fitness function f_i, the distribution of structures in the population m_i, at generation t. The value of ϕ for a schema H is calculated by adding the contributions of all the individual strings that are members of the schema H. P_s is a survival probability. The generalized schema theorem can alternatively be written in proportion form by dividing throughout by population size N as,

$$\langle P(H,t+1)\rangle \geq P(H,t)\gamma(H,m_i,f_i,t). \quad (4)$$

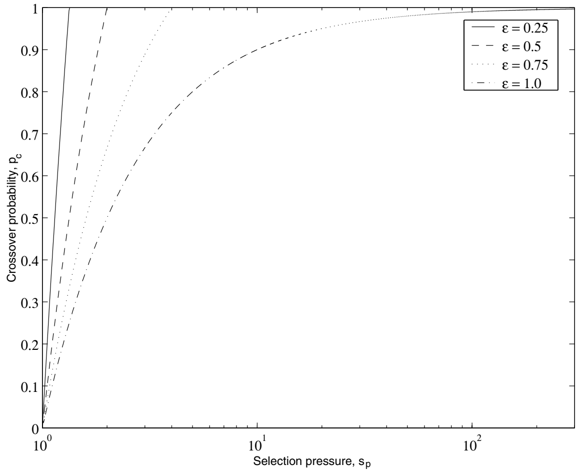


Figure 1: Limiting crossover probability p_c as a function of selection pressure s_p for different values of operator loss ϵ.

This theorem states that desirable schemata grow if $\gamma\,(H,m_i,f_i,t)\geq 1$. Although both the selection factor ϕ and the survival probability P_s are functions of the fitness function and the population, both quantities can be characterized more simply and are explained in the following section.

3 Designing for BB Growth

To employ the schema theorem in design, we simplify it by replacing ϕ with the selection pressure s_p, and parameterize the survival probability on an operator loss ϵ and the crossover probability p_c. The schema theorem can now be written as

$$\langle m\,(H,t+1)\rangle \geq m\,(H,t)\,s_p\,[1-p_c\epsilon]\,. \quad (5)$$

Then the desirable schema's grow if

$$s_p\,[1-p_c\epsilon]\geq 1. \quad (6)$$

Rearranging in terms of crossover probability p_c gives

$$P_c \leq \frac{1-s_p^{-1}}{\epsilon}. \quad (7)$$

The limiting p_c value is plotted as a function of selection pressure at different crossing losses in figure 1. We can see that even in the case of total loss of schema integrity, BB market share growth can still be ensured with reasonable combinations of sufficient selection pressure and diminished crossover probability. An interesting factor to note is that the schema theorem can always be satisfied with zero exchange — $p_c = 0$. However, in such a case the whole basis of operation principle of the GA with crossover is violated. This suggests that schema theorem must be obeyed, but that does not guarantee even a modicum of building block exchange, which is very important for a successful GA design.

To apply the BB design developed in this section in a real GA requires the consideration of selection pressure exerted by a given selection procedure. This issue is addressed in the next section for three different selection schemes.

4 Selection Schemes and Selection Pressure

We estimate the selection pressure s_p exerted in two phases, one early in the run and the other late in the run. The reason for doing so can be justified as follows: The initial generations are critical to the success of a GA run, because unless a schema (or its components) grow at the outset, its chances for success later on are quite poor. However, even if the conditions early in a GA push the best BBs fairly aggressively, but as the evolution wears on, even schemes with fairly steady drive toward convergence loose some of their initial punch. This might be a deal breaker, because the loss of selection pressure combined with high schema loss and fixed crossover probability might cause the evolution to stall before the best BBs dominate the population.

Here we consider the selection pressure exerted by three selection schemes, s-wise tournament, truncation, and proportionate selection procedures.

4.1 Tournament Selection

In s-wise tournament selection, s individuals are randomly drawn from a population (with or without replacement) and the best individual is selected. Assume that the initial proportion of superior BBs at time $t = 0$ is P_0 and is very small. The proportion of the best individuals under selection alone can be written as (Goldberg & Deb, 1991),

$$P_{t+1} = 1 - (1 - P_t)^s . \tag{8}$$

To consider the early performance of tournament selection, we calculate the selection ratio $\phi_t = P_{t+1}/P_t$ when $P_t \approx 0$.

$$\phi_t = P_t^{-1} \left[1 - (1 - P_t)^s\right] . \tag{9}$$

Recognizing that P_0 is small, and that $(1-x)^n \approx 1-nx$ for small x, we obtain

$$\phi_t \approx \frac{1 - (1 - sP_t)}{P_t} = s. \tag{10}$$

Thus early in the run when a good schema exists only in small proportions, the best schemas increase by a factor of s, the tournament size. This result is justified, since each individual participates in an average of s tournaments and the best individuals win all s of them. Equation 7 can be written for the case of tournament selection early in the run as

$$p_c \leq \frac{1 - s^{-1}}{\epsilon}. \tag{11}$$

As the proportion of a good schema becomes significant, the effective loss due to crossover reduces. This occurs due to the fact that for many crossover operators, a schema crossed with itself yields an instance of the same schema (Holland, 1975; Schaffer, 1987). The success probability, P_s, incorporating self-crosses can be written as

$$P_s = 1 - p_c' \left(1 - P_t\right) , \tag{12}$$

where $p_c' = p_c\epsilon$, and the term $1 - P_t$ is the result of self crossing — a schema contained in a string crossed with another string containing the schema is not disrupted regardless of the crossover point. The late behavior of the tournament selection is then given by

$$P_{t+1} = \left[1 - (1 - P_t)^s\right] \left[1 - p_c' \left(1 - P_t\right)\right] . \tag{13}$$

To perform a late analysis, we assume that the better structures have taken over more than $(1/s)^{\text{th}}$ of the population. Then the selection term $\left[1 - (1 - P_t)^s\right]$ approaches 1 and the late performance is described by

$$P_{t+1} = \left[1 - p_c' \left(1 - P_t\right)\right] . \tag{14}$$

The selection ratio, ϕ_t is given by

$$\phi_t = P_t^{-1} \left[1 - p_c' \left(1 - P_t\right)\right] . \tag{15}$$

It can be easily seen that $\phi_t \geq 1$ if (1) $0 < P_t \leq 1$, (2) $0 < p_c' \leq 1$, and (3) $s \left[1 - p_c'\right] \geq 1$, based on which the selection pressure and crossover probability are chosen for initial growth. Thus, by accounting for self-crossing we have shown that the subsequent effective slowing of convergence rate due to selection is more than overcome by the effective reduction in crossover probability.

4.2 Truncation Selection

Truncation selection is an ordinal selection scheme in which the top $1/s$ individuals in a population are given s copies each. If the proportion P_t of best individuals in a population is $1/s$ (early in the run), then the growth is geometric,

$$P_{t+1} = sP_t. \tag{16}$$

In other words, the selection rate $\phi_t = s$, which is the same as that for tournament selection. Once the proportion of good structures reaches or exceeds $1/s$ $\left(P_t \geq s^{-1}\right)$, the scheme saturates and the final proportion is,

$$P_{t+1} = 1. \tag{17}$$

Therefore the late performance is influenced only by the crossover operation and the proportion of good structures can be written as

$$P_{t+1} = \left[1 - p'_c\left(1 - P_t\right)\right]. \tag{18}$$

Thus truncation and tournament selection procedures have essentially similar late performance.

4.3 Proportionate Selection

To understand the early performance of proportionate selection, consider a highly simplified model. Consider two possible alternatives, 1 and 2, represented by objective function values f_1 and f_2 respectively with $f_2 > f_1$. Here the assumption is that alternative 1 is the average individual and alternative 2 is the best individual in the initial population. The proportion of alternative 2 can be tracked with the following difference equation,

$$P_{t+1} = \frac{f_2}{\overline{f}_t} P_t, \tag{19}$$

where $\overline{f}_t$ is the average function value at generation t, and is given by $\overline{f}_t = f_1\left(1 - P_t\right) + f_2 P_t$. Substituting this value in the above equation we get,

$$P_{t+1} = \frac{s}{(s-1)P_t + 1} P_t, \tag{20}$$

where $s = f_2/f_1$. The early performance, when there is a small proportion of good structures ($P_t \approx 0$), is given by $P_{t+1} \approx sP_t$. In other words, the selection rate for proportionate selection early in the run is given by $\phi_t = s$. This is similar to the early performance of tournament and truncation selection. However, unlike tournament and truncation selection schemes, the value of s will vary from problem to problem. It is very hard to know a priori whether an arbitrarily scaled problem will have adequate selection pressure. It is one of the reasons why ordinal selection procedures like tournament and truncation selection are preferred over proportionate selection (Baker, 1985). It is also one of the reasons for using some sort of scaling procedure along with proportionate selection.

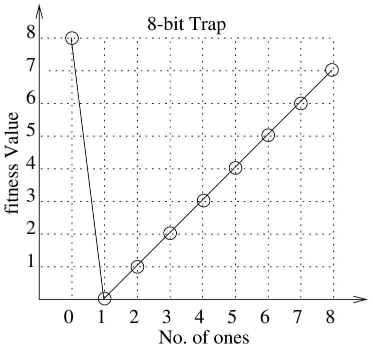


Figure 2: 8-bit trap function

Analysis of the late behavior of proportionate selection follows the similar procedure as that of the previous two selection schemes. Modifying equation 20 to incorporate self-crossing we get,

$$P_{t+1} = \frac{sP_t}{(s-1)P_t + 1}\left[1 - p'_c\left(1 - P_t\right)\right]. \tag{21}$$

Requiring the selection rate to be greater than or equal to one, we get

$$\frac{s}{(s-1)P_t + 1}\left[1 - p'_c\left(1 - P_t\right)\right] \geq 1. \tag{22}$$

Rearranging the above relation yields

$$p'_c \leq 1 - s^{-1}. \tag{23}$$

This model appears to show that proportionate selection would lead to full convergence in any situation for which the initial situation was favorable. However, it should be noted that s, defined as f_2/f_1, is not a constant during a run. In other words, both f_2 and f_1 can change in every generation, and also that the average fitness f_1 increases as a run goes on. This implies that the average fitness rises and the best BB's progress to dominate the population stalls. This phenomenon can be more accurately modeled by replacing the term s of equation 22 with a term where f_1 or the average fitness rises with increasing P_t. In the next section, we verify the theory with computational experiments.

5 Results

The theory developed in the above two sub-sections is verified with a computational experiment. Similar

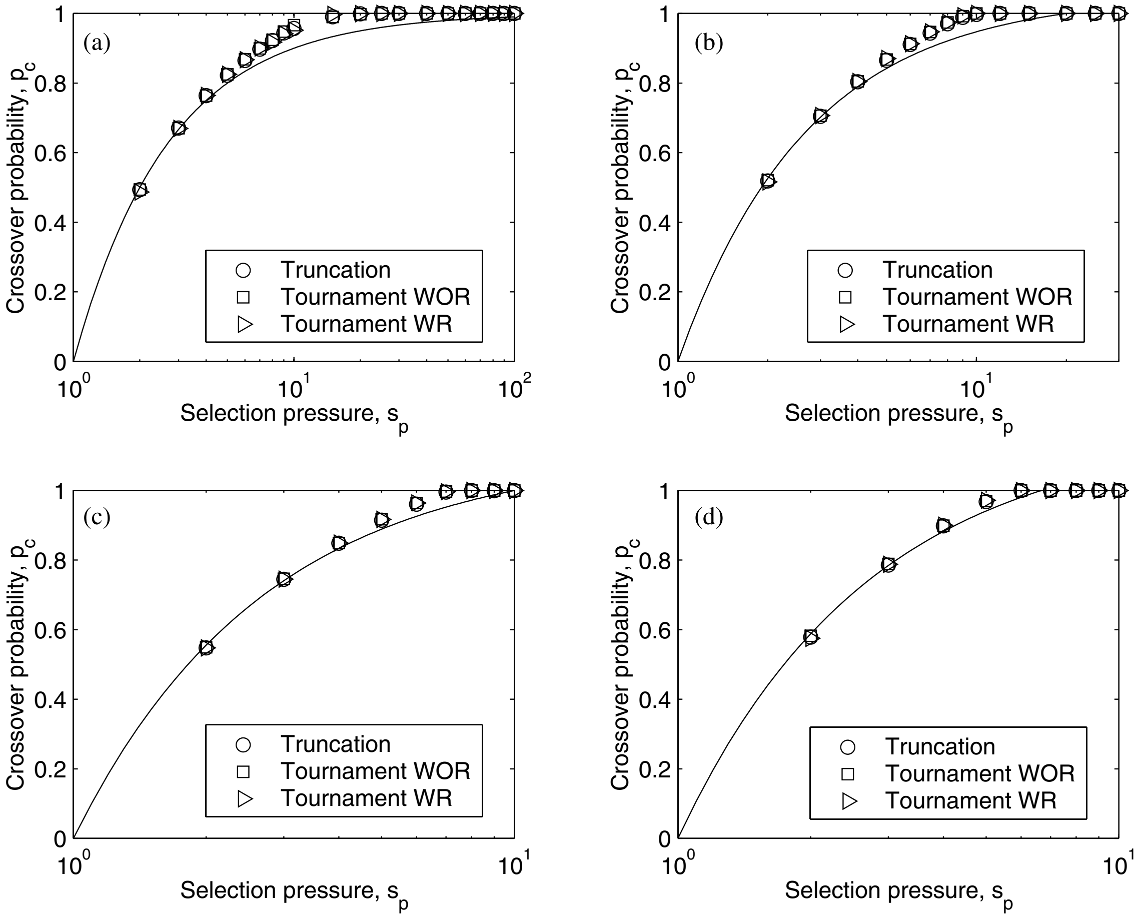


Figure 3: The experimental results showing the required crossover probability at a given selection pressure for tournament with or without replacement, and truncation selection at different disruption rates: (a) $\epsilon = 1.0$, (b) $\epsilon = 0.95$, (c) $\epsilon = 0.9$, and (d) $\epsilon = 0.85$.

control maps are reported elsewhere (Thierens, 1995), although the problem considered in the present study consists of a single building block and has a known schema disruption rate. We optimize a single 8-bit trap function (Goldberg, Deb, & Horn, 1992) (figure 2) with $f_8 = 7$, $f_0 = 8$, and trap break fitness $f_z = 0$ at $z = 1$. Single point crossover is used with either tournament selection (with and without replacement) or truncation selection and the disruption rate is fixed.

A trap function is used to make it difficult to find the best point (all-zeros, 00000000) and easy to find the second best (all-ones, 11111111). Experiments are run for specified s values to determine the crossover probability p_c when 25 independent runs successfully increase the proportion of the best strings for 10 consecutive generations. The population size taken was $N = 5000$. Bisection method was used to determine the p_c value within a tolerance of 10^{-5}. The results shown in figure 3 are average of 10 such independent runs of the bisection method. Both tournament and truncation selection agree with the design equation well at different disruption rates, $\epsilon = 0.85$, 0.9, 0.95, and 1.0.

The theory on proportionate selection is verified using stochastic universal selection. In these runs, neither scaling nor ranking is employed and hence the selection pressure cannot be manipulated independent of the problem scaling. This results in three cases: (1) convergence to the best point (All-zeros, 00000000), (2) stall of the best point, at some proportionate value, and (3) mis-convergence to the second best point (all-ones, 11111111). These three cases are exemplified in figures 4(a)–(d). The results shown are based on 25 successes in 25 trials in a population of 5000.

Figure 4(a) shows a successful takeover of the population by the best point. Initially, the inferior point (all-ones) grows faster, but due to low crossover probability

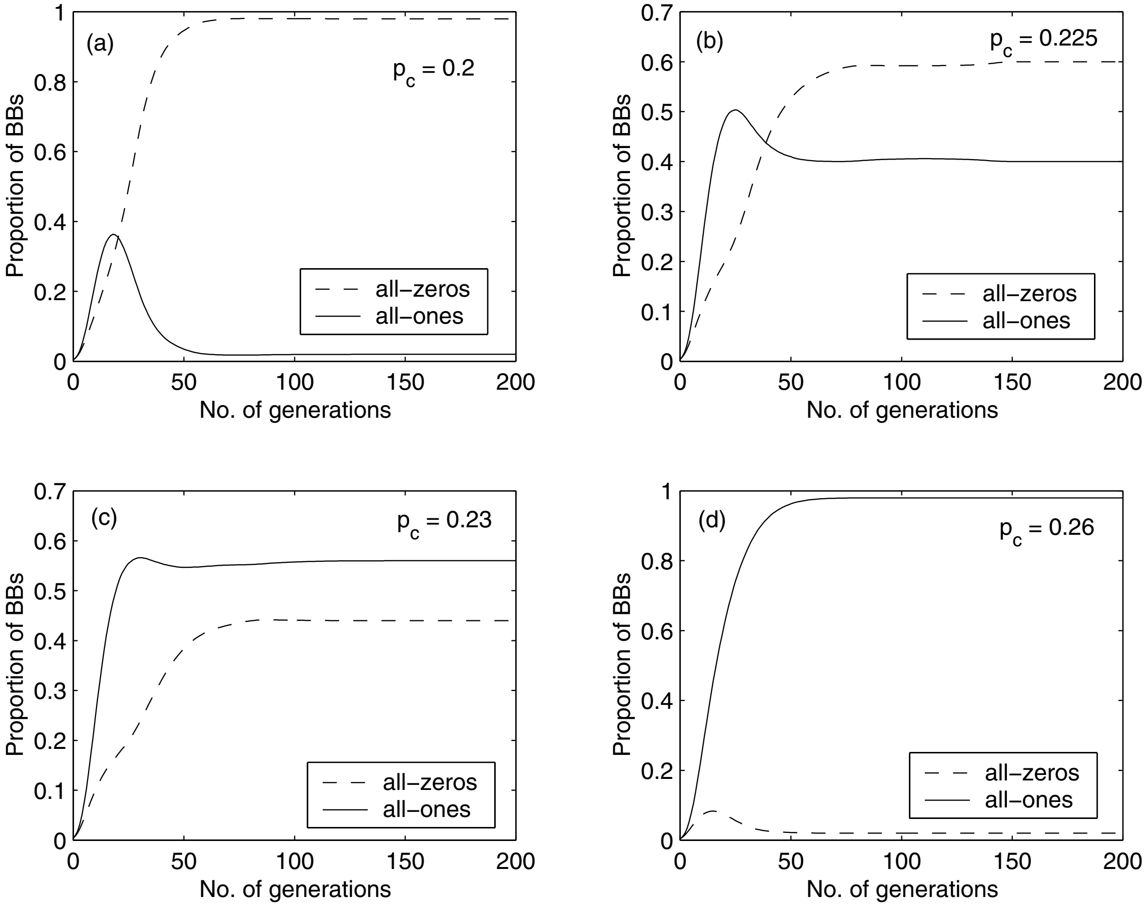


Figure 4: The proportion of all-ones points and all-zeros points is plotted versus generation number at different crossover probabilities. The results are averaged over 25 independent runs.

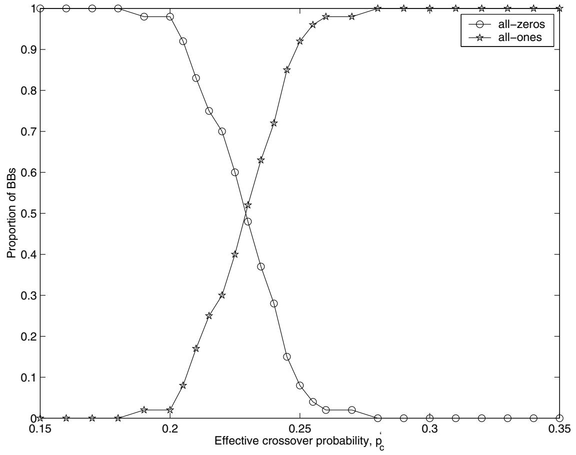


Figure 5: The proportion of all-ones points and all-zeros points when stall occurs is plotted versus crossover probability. The results are averaged over 100 independent runs.

($p_c = 0.2$), the superior point is able to completely take over the population. Figures 4(b)–(c) demonstrates the stall of all-zeros proportion at crossover probabilities, $p_c = 0.225$, and $p_c = 0.23$ respectively. Finally, figure 4(d) indicates failure of the best point to takeover the population at $p_c = 0.26$. The proportion of best and the second best point when stall occurs is plotted against crossover probability in figure 5. The results are averaged over 100 independent trials. The plot shows that at low crossover probabilities the best point successfully takes over the population, at high crossover probabilities the best point fails to sustain the market share increase, and at intermediate crossover probability values the best point stalls at some intermediate proportion value. However, it should also noted that we can have either a total success or total failure in take over of the population by the best point. Examples of the same phenomena are shown in figures 6(a)–(d). In all these cases the proportion of best point stalls for a long time (30–70 generations) and then either it succeeds or fails to takeover

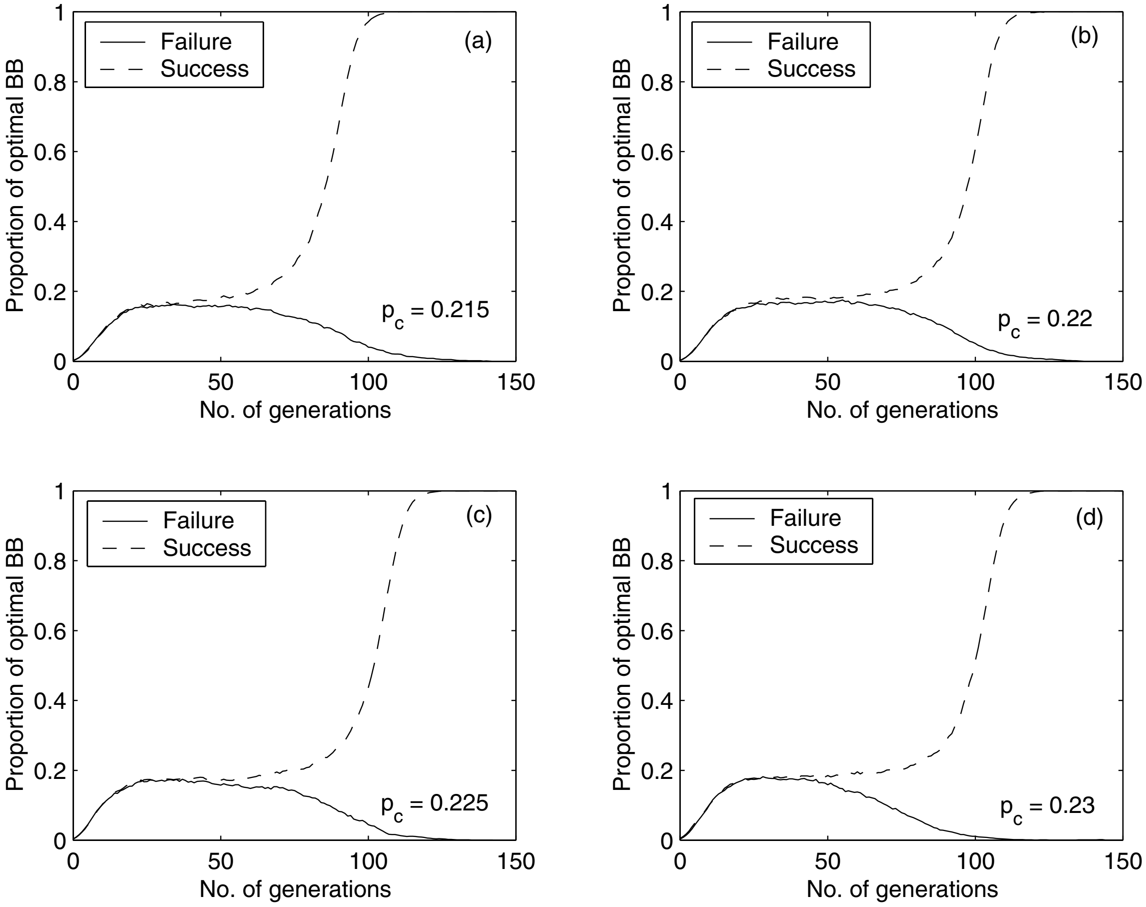


Figure 6: Demonstration of success and failure after intermediate stall at different crossover probabilities (a) $p_c = 0.215$, (b) $p_c = 0.22$, (c) $p_c = 0.225$, and (d) $p_c = 0.23$. The results are of a single run

the population. The results shown are at crossover probabilities of 0.21, 0.215, 0.22, and 0.225. These results indicate the reason for preferring pushier schemes like tournament and truncation selection over proportionate selection.

6 Conclusion

This study clearly shows that the schema theorem works with different selection schemes and genetic operators and that it must be obeyed. The schema theorem provides a good bounding advice on how to assure the growth of good subsolutions and to sustain the growth to takeover the population. Employing early and late analysis, it has been demonstrated that one can set the GA parameters, the selection pressure, s and the crossover probability p_c for tournament and truncation selection schemes to obey schema theorem. Unscaled proportionate schemes have a tendency to stall, indicating that proportionate selection schemes though useful when applied with scaling procedures or niching, should not be generally used without such augmentation. The fact that schema theorem can be satisfied with a crossover probability of zero suggests that schema theorem does not consider the positive effects of crossover, the exchange of BBs that is at the heart of GA mechanics.

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 Martin Pelikan, Ravi Srivastava and Abhishek Sinha for careful readings of this paper and a number of comments.

References

Baker, J. (1985). Adaptive selection methods for genetic algorithms. In Grefenstette, J. (Ed.), *Proceedings of the International Conference on Genetic Algorithms and Their Applications* (pp. 101–111). Hillsdale, NJ: Lawrence Erlbaum Associates.

Baker, J. (1987). Reducing bias and inefficieny in the selection algorithm. In Grefenstette, J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 14–21). Hillsdale, NJ: Lawrence Erlbaum Associates.

De Jong, K. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* (university microfilms no. 76-9381), University of Michigan, Ann Arbor.

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

Goldberg, D., & Deb, K. (1991). A comparitive analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, Volume 1 (pp. 69–93). San Mateo, CA: Morgan Kaufmann.

Goldberg, D., Deb, K., & Clark, J. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D., Deb, K., & Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. In Manner, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature*, Volume 2 (pp. 37–46). UK: Elsevier Science Publishers.

Goldberg, D., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530.

Goldberg, D., Sastry, K., & Latoza, T. (2001). On the supply of building blocks. In *Proceedings of Genetic and Evolutionary Computation Conference.* San Francisco, CA: Morgan Kaufmann.

Grefenstette, J., & Baker, J. (1989). How genetic algorithms work: A critical look at implicit parallelism. In Schaffer, J. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 20–27). San Mateo, CA: Morgan Kaufmann.

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

Muhlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Schaffer, J. (1987). Some effects of selection procedures on hyperplane sampling by genetic algorithms. In Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing* (pp. 89–103). Los Altos, CA: Morgan Kaufmann.

Thierens, D. (1995). *Analysis and Design of Genetic Algorithms.* Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.

On The Supply Of Building Blocks

David E. Goldberg, Kumara Sastry, and Thomas Latoza
Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
104 S. Mathews Ave, Urbana, IL 61801
{deg,ksastry,latoza}@uiuc.edu

Abstract

This study addresses the issue of building-block supply in the initial population. Facetwise models for supply of a single building block as well as for supply of all schemata in a partition have been developed. An estimate for the population size required to ensure the presence of all raw building blocks has been derived using these facetwise models. The facetwise models and the population-sizing estimate are verified with computational results.

1 Introduction

The importance of building blocks (BBs) and their role in the working of GAs have long been recognized (Holland, 1975; Goldberg, 1989a). Elsewhere the problem of successful design of a selectorecombinative GA has been decomposed into six subproblems (Goldberg, Deb, & Clark, 1992): (1) Know what GAs are processing - building blocks (BBs), (2) ensure an adequate initial supply of raw BBs, (3) Ensure growth of superior BBs, (4) ensure the mixing of BBs, (5) ensure good decisions among competing BBs, and (6) solve problems with bounded BB difficulty. One of the essential steps towards successful design of a GA is making sure that the GA is well supplied with a sufficient number of the BBs required to solve a given problem. There are two approaches to address the BB supply question, a spatial and a temporal one. The spatial approach estimates the population size required to ensure diversity and the number of BBs present in the initial population. The temporal approach assumes the existence of a mutation or other diversity generator to return sufficient BB diversity on an appropriate time scale. In this study we restrict ourselves to BB supply in selectorecombinative GAs, and hence address the spatial approach.

The objective of this study is to develop a facetwise model for supply of BBs and to estimate the population size required to guarantee the presence of all raw BBs in the initial population. Though, ensuring BB growth supersedes BB supply in the subsequent population, BB growth will be extremely difficult if BB supply is not ensured. While decision making governs population sizing usually, it is sometimes governed by BB supply. In such cases a facetwise model of BB supply is necessary for ensuring a successful GA design. It should be noted that there exist sophisticated models that combine the requirements of supply and decision making in a single model (Harik, Cantu-Paz, Goldberg, & Miller, 1997). These complex models are composed of simpler models, and blend the effects of those simple models. Therefore, developing simple facetwise models are useful for obtaining insight in more complex models. Here, we consider the BB supply question in isolation and derive a facetwise model using some straightforward probabilistic calculations. In this study we restrict ourselves to strings with fixed length, fixed alphabet cardinality, and fixed BB size. We start with a brief review of past work in this area. Then two facetwise models of the probability of BB supply are derived. Subsequently, an estimate of population size is presented by using the second facetwise model.

2 Brief Literature Review

A full review of past work on the supply of BBs is beyond the scope of this paper and hence a brief review is presented. Holland (1975) was the first to address the issue of BB supply. He estimated the number of BBs that receive at least a specified number of trials using Poisson distribution. A later study (Goldberg, 1989b) calculated the same quantity more exactly using binomial distribution and used those calculations to study their effects on population sizing in serial and parallel computation. Recently Reeves (1993) proposed a population-sizing model for supply of BBs with fixed cardinality. However, he only considered BBs of unit size. Holland (1973, 1975) addressed the issue of decision making using the analogy of a two-armed bandit problem. De Jong (1975) incorporated the effects of noise in the decision process and proposed a population-sizing model based on the signal and noise characteristics of the problem. Goldberg and Rudnick (1991) developed a population-sizing model based on variance of fitness. A subsequent work (Goldberg, Deb, & Clark, 1992) developed a decision-based model to estimate population size to minimize decision errors. Their model is based on deciding correctly between the best and second best BBs in a partition in the presence of noise arising from other partitions. Harik, Cantu-Paz, Goldberg, and Miller (1997) refined this model by incorporating cumulative effects of decision making over a GA run. They also incorporated the effects of initial BB supply in their population-sizing model.

3 Facetwise Model for the Supply of a Single BB

Consider the probability of having a single k-position schema, p_k represented by one or more structures in a randomly generated population of size n. Let the population consist of strings of cardinality χ, k be the order of the schema. p_k is given by

$$p_k = 1 - \left[1 - \left(\frac{1}{\chi^k}\right)\right]^n . \qquad (1)$$

Using the approximation, $(1 - r/n)^n \approx e^{-r}$, and recognizing that this approximation is sufficiently accurate even for modest values of n, we can write

$$p_k = 1 - \exp\left(-\frac{n}{\chi^k}\right) . \qquad (2)$$

The above equation is a simplified expression for the probability of having one or more successes at given schema. This model is compared to empirical results with alphabet cardinality of 2, 3, 4 and 5.

Figures 1(a), 1(b), 1(c) and 1(d) depict the proportion of runs out of 1000 trials for which at least one copy of a particular BB was present, for alphabet cardinalities of $\chi = 2$, $\chi = 3$, $\chi = 4$, and $\chi = 5$ respectively. The empirical results agree more accurately with the analytical results as the population size, n increases and as the term χ^k increases. An extension of this facetwise model to incorporate success at all schemas of a partition is presented in the next section.

4 Facetwise Model of Supply for Partition Success

When solving real-world problems, one does not have prior knowledge about a particular schema being superior to others in a partition. Hence it is necessary to ensure that all competing schemata in a partition are present. The decision process would then be able to consider all the relevant alternative schemata. Therefore in this section we extend the model developed in the previous section to ensure the presence of at least one copy of all the competing schemata in a partition. For example, if a particular problem requires the last two bits of a four-bit string to be evaluated jointly ($**$ff), we should ensure that all schemata in the partition ($**00$, $**01$, $**10$, $**11$) be present in the initial population.

We first derive an exact model to predict the probability of having at least one copy of all the competing schemata in a partition for the case k = 1 and alphabet cardinalities $\chi = 2$, 3, and 4. Then a generalized model of this probability for any schema-order and alphabet cardinality is presented. First considering the case $\chi = 2$, for a population of size n, the probability of having at least one copy of $1*$ and $0*$, p_s can be written as: $p_s = 1$ - probability of all the individuals being either $1*$ or $0*$. That is,

$$\begin{aligned} p_s &= 1 - \left(\frac{1}{2^n} + \frac{1}{2^n}\right), \\ &= 1 - \frac{1}{2^{n-1}}. \end{aligned} \qquad (3)$$

For $\chi = 3$, p_s is the complement of the probability that none of the individuals have at least one schema ($0*$, $1*$, $2*$). The number of possible ways of not having at least one of the schemas in a given partition, n_f, can be written as

$$n_f = 3 + 3\sum_{i=1}^{n-1} \binom{n}{i} . \qquad (4)$$

The first term represents the possibility of having identical schema in all individuals and the second term represents all possibilities of having two schemas. Using the binomial theorem, $\sum_{i=0}^{n} \binom{n}{i} = 2^n$, we can write

$$n_f = 3 + 3\left(2^n - 2\right) = 3\left(2^n - 1\right) . \qquad (5)$$

It can be easily seen that the total number of possible ways of schemas being present in the population is $\chi^n = 3^n$. The probability of success, p_s is given by

$$p_s = 1 - \frac{n_f}{3^n},$$

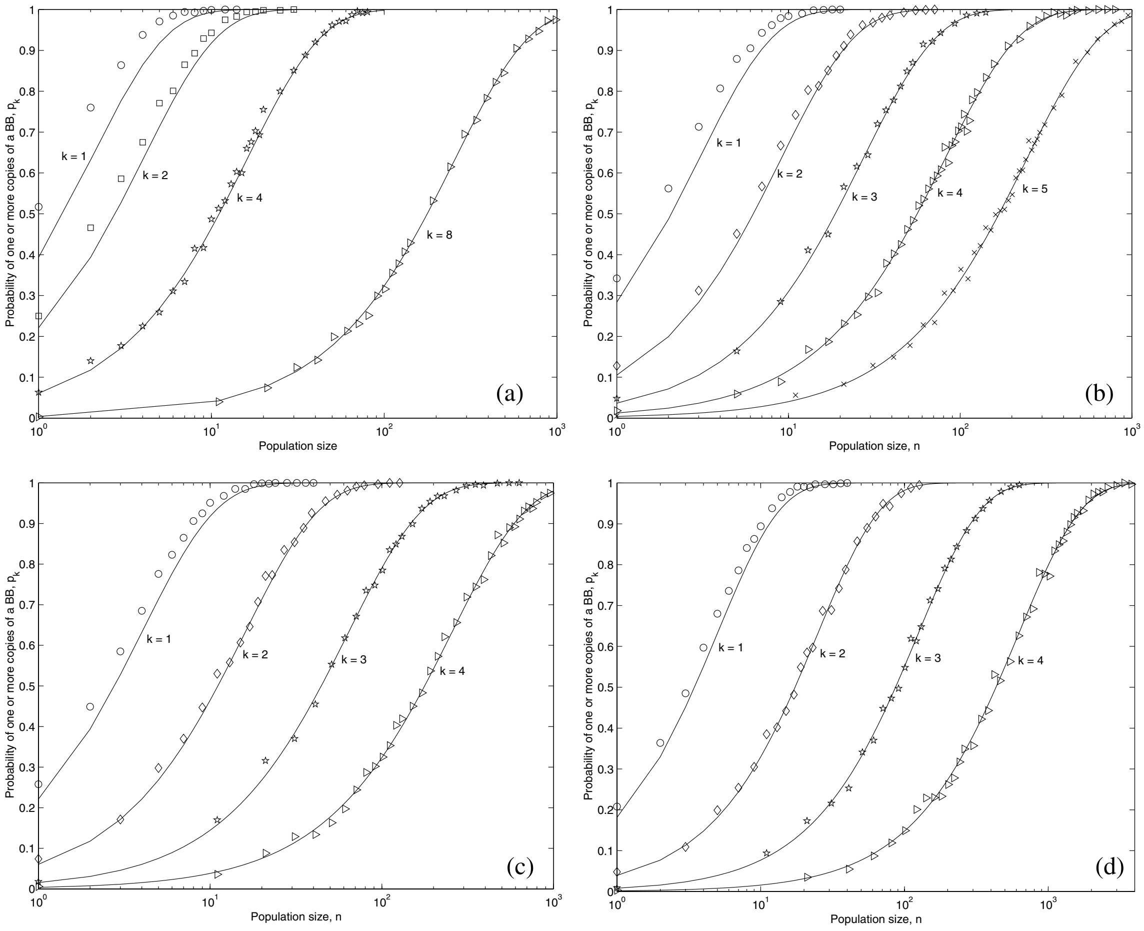


Figure 1: Verification of the facetwise model for a single BB supply (equation 2) with empirical results for different schema order values, k, and alphabet cardinalities, χ, as a function of population size, n. The empirical results depict the proportion of runs having at least one copy of a particular schemata out of 1000 trials. Cardinality of the string (a) $\chi = 2$, (b) $\chi = 3$, (c) $\chi = 4$, and (d) $\chi = 5$. Equation 1 matches the experimental results exactly and the agreement between equation 2 and experimental results increases with n and χ^k.

$$= \quad 1 - \frac{2^n - 1}{3^{n-1}}. \qquad (6)$$

Similarly, for $\chi = 4$, the number of possible ways of not having at least one of the schemata in a partition, n_f is given by

$$\begin{aligned} n_f \quad = \quad & \binom{4}{1} + \binom{4}{2} \sum_{i=1}^{n-1} \binom{n}{i} + \binom{4}{3} \\ & \sum_{i=1}^{n-2} \left[\binom{n}{i} \sum_{k=1}^{n-i-1} \binom{n-i}{k} \right]. \qquad (7) \end{aligned}$$

Some straightforward simplification of the above equation yields,

$$n_f = 8 - 2^{n+1} + 8n + 4 \sum_{i=1}^{n-2} \left[\binom{n}{i} 2^{n-i} \right]. \qquad (8)$$

Using the binomial theorem, we can equate

$$\sum_{i=1}^{n-2} \left[\binom{n}{i} 2^{n-i} \right] = 3^n - 2^n - 1 - 2n.$$

Substituting the above result in equation 8, we get

$$n_f = 4 \left(3^n - 3 \cdot 2^{n-1} + 1 \right). \qquad (9)$$

The probability of success, p_s is given by

$$\begin{aligned} p_s \quad & = \quad 1 - \frac{n_f}{4^n}, \\ & = \quad 1 - \frac{3^n - 3 \cdot 2^{n-1} + 1}{4^{n-1}}. \qquad (10) \end{aligned}$$

In general, for a schema-order value of k and alphabet cardinality of χ, the number of possible ways of not having at least one of the schema in a given partition can be written as

$$n_f = \sum_{i=1}^{n_k-1} (-1)^{i-1} \left(\begin{array}{c} n_k \\ i \end{array} \right) (n_k - i)^n . \qquad (11)$$

where $n_k = \chi^k$. Therefore the probability of success of having at least one copy of all schemata in a given partition is given by

$$p_s = 1 - \frac{1}{n_k^n} \left[\sum_{i=1}^{n_k-1} (-1)^{i-1} \left(\begin{array}{c} n_k \\ i \end{array} \right) (n_k - i)^n \right] . \quad (12)$$

We can easily see that when $n > n_k$, the above equation can be approximated as

$$p_s \approx 1 - n_k \exp\left(-\frac{n}{n_k}\right) . \qquad (13)$$

The exact model becomes complicated for higher values of schema order and does not give us useful insight. Hence we may derive a simpler form by assuming that the schema partition success values are independent. Then the probability of at least one success at each of the $n_k = \chi^k$ partitions, p_s is given by $p_s = p_k^{n_k}$. Using equation 2 we get

$$p_s = \left[1 - \exp\left(-\frac{n}{n_k}\right) \right]^{n_k} . \qquad (14)$$

Again using the approximation, $(1 - r/n)^n \approx e^{-r}$, results in the following relation,

$$p_s = \exp\left[-n_k \exp\left(-\frac{n}{n_k}\right) \right] . \qquad (15)$$

When $n > n_k$, using the approximation $(1-x)^n \approx 1 - nx$ for small values of x, equation 14 can be written as

$$p_s \approx 1 - n_k \exp\left(-\frac{n}{n_k}\right) . \qquad (16)$$

Equations 13, and 16 show that the approximate model agrees with the exact model of the BB success probability for higher population sizes. The simplified model of equation 15 is compared to the exact result of equation 12 and empirical cases for $k = 1$ in figure 2. The empirical results match the exact model very accurately. The approximate model is a conservative estimate of the probability and it agrees well with empirical result at higher population sizes.

The above approximate model (equation 14) is verified with empirical results in figures 3(a)–(d). The plots compare the proportion of runs out of 1000 trials that have at least one copy of all members of a particular schema partition for alphabet cardinality of $\chi = 2$, 3, 4, and 5. As seen earlier we can see that the agreement between the empirical results and the analytical model increases with the increase in the schema order, k, and the population size, n.

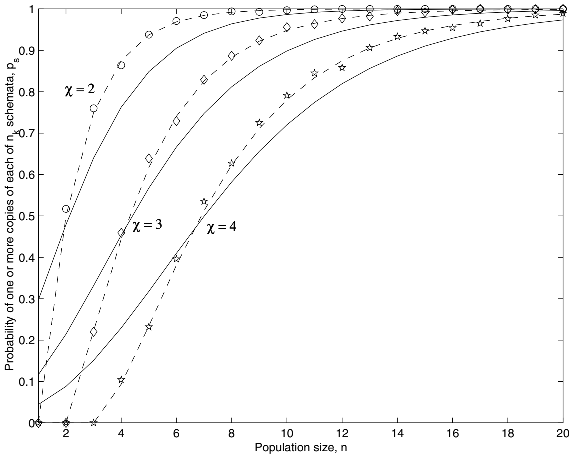


Figure 2: Comparison of the exact (equation 12) and approximate (equation 14) models for BB partition success with empirical results. The empirical results depict proportion of runs having at least one copy of all members of a particular schema partition out of 1000 trials for schema order value, $k = 1$ as a function of population size, n, for different alphabet cardinality, χ. The solid line represents equation 14, and the dashed line represents equation 12. The agreement between equations 12 and 14 increases as n increases.

5 Population Size for BB Supply

The facetwise model derived in the previous section will be rearranged in this section to estimate the population size required to ensure the presence of all BBs of a partition for problems of varying BB size, k, count, m, and alphabet cardinality, χ. Assuming that we can tolerate a probability α of not having all BBs in a given partition, and setting p_s to $1 - \alpha$, we can rewrite equation 15,

$$1 - \alpha = \exp\left[-n_k \exp\left(-\frac{n}{n_k}\right) \right] . \qquad (17)$$

Taking logarithm on both sides and using the approximation $\log(1 - x) \approx -x$, for small values of x, gives

$$\alpha = n_k \exp\left(-\frac{n}{n_k}\right) . \qquad (18)$$

Solving the above equation for n yields

$$n = n_k \left(\log n_k - \log \alpha \right) . \qquad (19)$$

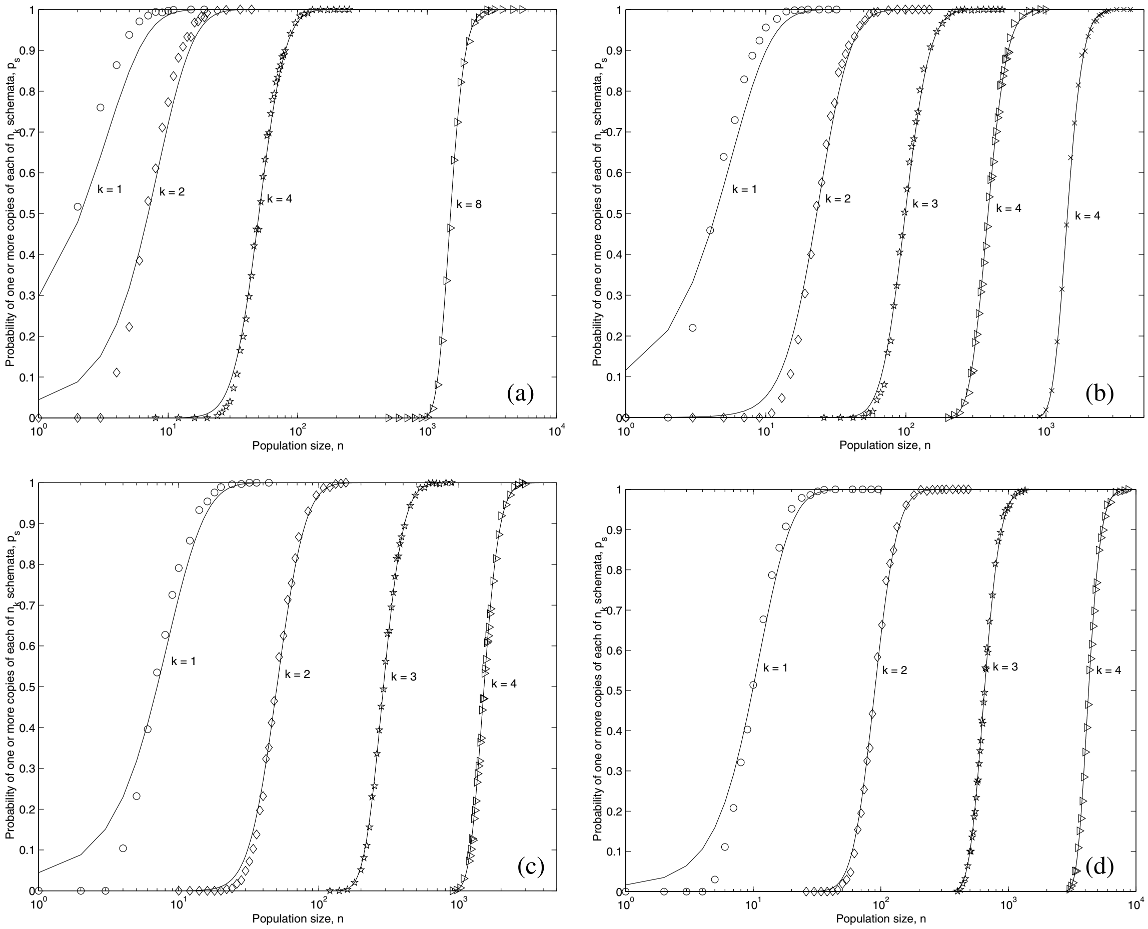


Figure 3: Verification of the model for BB partition success (equation 14) with empirical results for different schema order values, k, and alphabet cardinalities, χ, as a function of population size, n. The empirical results depict the proportion of runs having at least one copy of all members of a particular schema partition out of 1000 trials. Alphabet cardinality (a) $\chi = 2$, (b) $\chi = 3$, (c) $\chi = 4$, and (d) $\chi = 5$. Equation 12 matches the empirical results exactly and the agreement between equation 14 and experimental results increases with n and χ^k.

Using the definition of n_k we get,

$$n = \chi^k \left(k \log \chi - \log \alpha\right). \tag{20}$$

This relation yields the estimate of the population size required to ensure the presence of all BBs of a partition. We can simplify this relation further if we assume that the supply error is inversely proportional to the number of BBs, m, that is, $\alpha = 1/m$. Then the equation may be rewritten as

$$n = \chi^k \left(k \log \chi + \log m\right). \tag{21}$$

It can be easily seen that the above result can also be obtained from equations 13, and 16. The above population-sizing equation is an upper bound on the population size required to ensure the presence of all the BBs in the initial population. This model is verified with empirical results in figures 4(a)–(d). The plots depict population size vs the number of building blocks, m, at different k values and different alphabet cardinality χ values. It can be easily seen from the plots that the model results agree with empirical results. The results shown are averaged over 100 independent runs, with all 100 runs yielding a supply error of less than or equal to $1/m$.

The population-sizing equation has two asymptotic cases. One when a problem is relatively large with respect to its complexity — when $m \gg \chi^k$ — and the other in which the problem is relatively complex with respect to its size — when $\chi^k \gg m$. In the first case, the population size required for ensuring the pres-

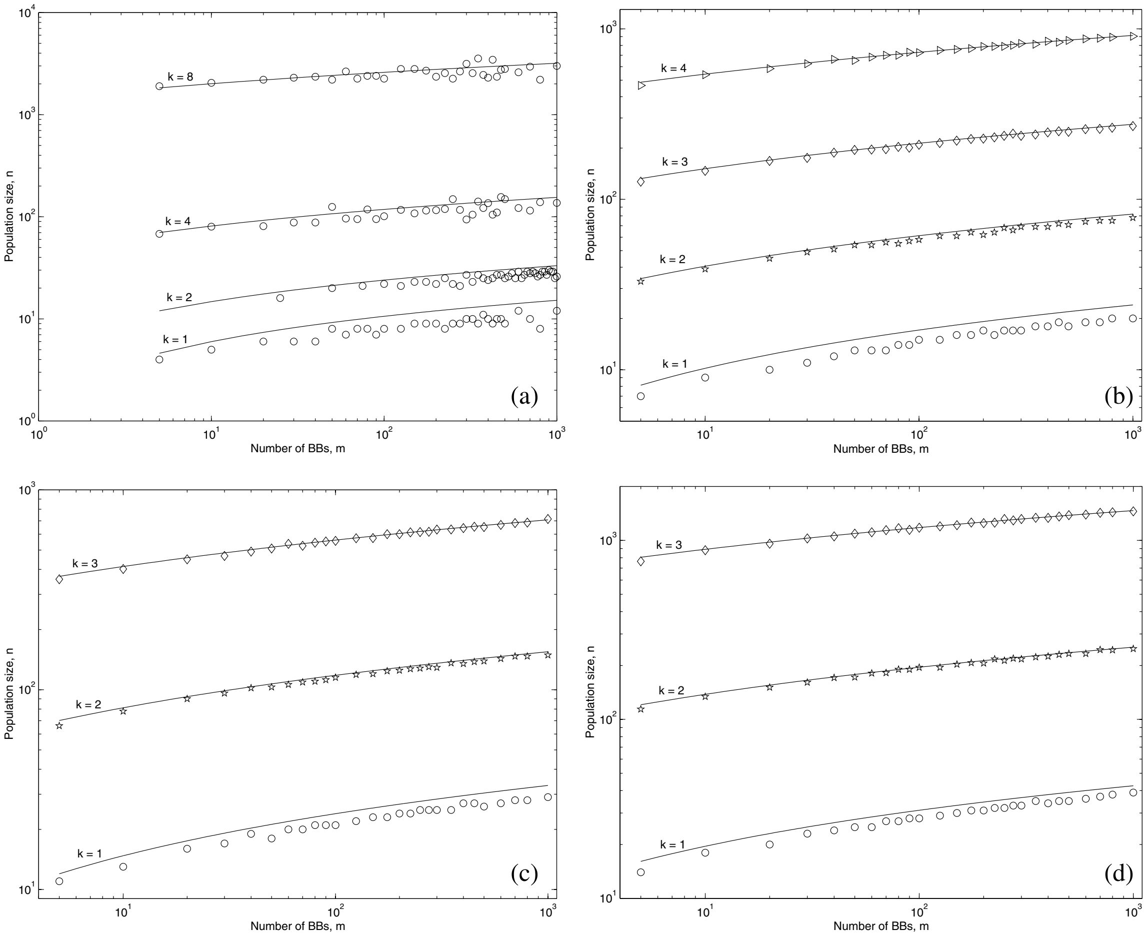


Figure 4: Verification of the population-sizing model for BB supply (equation 21) with empirical results for different partition sizes, k, and alphabet cardinalities, χ, as a function of number of BBs, m. The empirical results depict the population size required to represent all schemas in a partition of size k at a supply error of $1/m$. The experimental results are averaged over 100 runs. Alphabet cardinality (a) $\chi = 2$, (b) $\chi = 3$, (c) $\chi = 4$, and (d) $\chi = 5$. The agreement between equation 21 and experimental results increases with n and χ^k.

ence of all the schemas in a partition is $O(\chi^k \log m)$. In the later case, the population size required on BB supply grounds is $O(k\chi^k \log \chi)$. In both cases, the supply population-sizing is less than that required on decision-making grounds (Goldberg, Deb, & Clark, 1992; Harik, Cantu-Paz, Goldberg, & Miller, 1997).

6 Conclusions

In this paper, a detailed analysis of BB supply in the initial population, one of the six essential steps for a successful GA design, has been presented. Two facetwise models are derived, one for ensuring supply of a single schemata in a partition, and the other for ensuring supply of all competing schemata in a partition. The latter model has been employed to estimate the population size required to ensure the presence of at least one copy of all raw BBs of a partition in the initial population. The population-sizing model indicates that for large easy problems the population size required on BB supply grounds is $O(\chi^k \log m)$, and for relatively complex problems the population size is $O(k\chi^k \log \chi)$.

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 Ravi Srivastava and Abhishek Sinha for careful readings of this paper and for their comments.

References

De Jong, K. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan, Ann-Arbor, MI.

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

Goldberg, D. (1989b). Sizing populations for serial and parallel genetic algorithms. In Schaffer, J. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 70–79). San Mateo, CA: Morgan Kaufmann.

Goldberg, D., Deb, K., & Clark, J. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D., & Rudnick, M. (1991). Genetic algorithms and the variance of fitness. *Complex Systems*, *5*(3), 265–278.

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.

Holland, J. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, *2*(2), 88–105.

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

Reeves, C. (1993). Using genetic algorithms with small populations. In *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 92–99). San Mateo, CA: Morgan Kaufmann.

Prüfer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search

Jens Gottlieb
SAP AG
Neurottstr. 16
69190 Walldorf, Germany
jens.gottlieb@sap.com

Bryant A. Julstrom
Department of Computer Science
St. Cloud State University
720 Fourth Avenue South
St. Cloud, MN 56301, USA
julstrom@eeyore.stcloudstate.edu

Günther R. Raidl
Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstr. 9–11/1861
1040 Vienna, Austria
raidl@ads.tuwien.ac.at

Franz Rothlauf
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
104 S. Mathews Avenue
Urbana, IL 61801, USA
rothlauf@uni-bayreuth.de

Abstract

The most important element in the design of a decoder-based evolutionary algorithm is its genotypic representation. The genotype-decoder pair must exhibit efficiency, locality, and heritability to enable effective evolutionary search. Prüfer numbers have been proposed to represent spanning trees in evolutionary algorithms. Several researchers have made extravagant claims for the usefulness of this coding, but others have pointed out that Prüfer numbers, though concise and easy to decode, lack the essential properties of locality and heritability. This conflict motivates our study. We examine the properties of Prüfer numbers and compare Prüfer numbers with other codings in evolutionary algorithms for four problems that involve spanning trees. Our conclusion is definite: Prüfer numbers cause poor performance in evolutionary algorithms and should be avoided.

1 Introduction

An evolutionary algorithm (EA) maintains a population of data structures that represent candidate solutions to a problem. In a *decoder-based* EA, each data structure can be thought of as providing instructions to a decoder that builds the solution the structure represents. The data structure is the solution's *genotype*; the decoded solution is its *phenotype*.

In EAs for problems of combinatorial optimization, the decoder can consider problem-specific information such as constraints; the EA's author then need not design complex evolutionary operators or penalty functions. The decoder should be fast, and the coding implemented in the genotype-decoder pair should exhibit *locality* and *heritability*: small changes in genotypes should correspond to small changes in the solutions they represent, and solutions generated by crossover should combine features of their parents.

Decoder-based EAs have been applied to problems that search spaces of spanning trees. Finding an unconstrained minimum spanning tree is easy, but many problems involving spanning trees are computationally hard and thus suitable targets for heuristics such as EAs. *Prüfer numbers* offer a deceptively elegant coding of spanning trees whose decoding algorithm is found in a constructive proof of Cayley's formula.

This formula identifies the number of unconstrained spanning trees in a complete undirected graph on n nodes as n^{n-2} (Cayley, 1889; Even, 1973, pp.98–106). Prüfer (1918) described a one-to-one mapping between spanning trees on n nodes and strings of $n-2$ node labels. Conventionally, integers label the nodes, and the strings of labels are called Prüfer numbers.

Prüfer numbers thus encode spanning trees, and researchers have used Prüfer numbers in EAs for problems that search spaces of spanning trees. These have included the probabilistic minimum spanning tree problem (Abuali et al., 1994), the degree-constrained minimum spanning tree problem (Zhou and Gen, 1997), the time-dependent minimum spanning tree problem (Gargano et al., 1998), the fixed-charge transportation problem (Li et al., 1998) and a bicriteria version of it (Gen and Li, 1999), and a multi-objective network design problem (Kim and Gen, 1999). Some authors have made extravagant claims for the efficacy of this coding, including Kim and Gen, who wrote

> The Prüfer number is very suitable for encoding a spanning tree, especially in some research fields, such as transportation problems, minimum spanning problems, and so on.

Such claims are in error. In general, Prüfer numbers do not support effective evolutionary search. Alternate codings of spanning trees consistently provide better results, as we show in four examples.

This paper describes Prüfer numbers and their properties and compares Prüfer numbers with other codings of spanning trees in EAs for four problems: the degree-constrained spanning tree problem, the communication spanning tree problem, the rectilinear Steiner problem, and the fixed-charge transportation problem. In every case, evolutionary search is more effective with other codings than with Prüfer numbers.

2 The Prüfer Number Representation

This section describes the algorithm that decodes a Prüfer number to a spanning tree, examines the locality and heritability of the Prüfer coding under conventional evolutionary operators, and finds that the region of the search space where a Prüfer-coded EA might perform well is vanishingly small.

2.1 Decoding Prüfer Numbers

Let $a_1 a_2 \cdots a_{n-2} \in \{1, \ldots, n\}^{n-2}$ be a Prüfer number. In the corresponding spanning tree, each node's degree is one more than the number of times the node's label appears. To identify the spanning tree's edges:

1. Scan the Prüfer number to identify each node's degree. Initialize a variable i to 1.
2. Find the node v of degree 1 with the smallest label. (v, a_i) is a spanning tree edge.
3. Decrement the degrees of v and a_i; increment i.
4. Repeat steps (2) and (3) until all nodes have degree 0, except two with degree 1. These form the spanning tree's last edge.

An efficient implementation of this algorithm uses a priority queue implemented in a heap to hold the nodes of degree 1. The algorithm's time is then $O(n \log n)$. Figure 1 shows a Prüfer number of length four and the tree on six nodes to which it decodes.

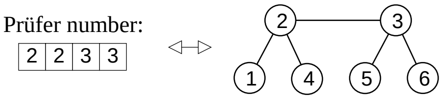


Figure 1: A spanning tree and its Prüfer number.

Several features of Prüfer numbers suggest that they might support efficient evolutionary search of spaces of spanning trees. They can be decoded quickly; it is easy to generate random Prüfer numbers by choosing $n-2$ times from n node labels; and Prüfer numbers support conventional evolutionary operators like k-point crossover and position-by-position mutation. However, Prüfer numbers' poor locality and heritability are not conducive to evolutionary search, as the remainder of this section demonstrates.

2.2 Locality

A coding has high locality if mutating a genotype changes the corresponding phenotype only slightly. Several researchers, including Palmer and Kershenbaum (1994) and Rothlauf and Goldberg (2000), have pointed out the poor locality of Prüfer numbers with respect to conventional position-by-position mutation.

In general, Prüfer numbers related by such mutations do not represent similar spanning trees. For example, changing the last digit in the Prüfer number of Figure 1 from 3 to 1 yields 2231, which decodes to the edges (2,4), (2,5), (3,2), (1,3), and (1,6). Only two of the original tree's five edges remain.

Some Prüfer numbers do have high locality. A star is a spanning tree in which every node but one is a leaf. A star on n nodes has $(n-1)(n-2)$ neighbors, obtained by replacing one of its edges with another feasible edge. In a star's Prüfer number, all the symbols are the same. A star's neighbors are represented by the neighbors of its Prüfer number, obtained by changing one of the digits; these neighbors also number $(n-1)(n-2)$. For stars, the genotypic and phenotypic neighborhoods coincide, and locality is maximal.

This seems auspicious, but Prüfer numbers' localities vary with the shapes of the trees they represent. A list is a spanning tree with two leaves and $n-2$ nodes of degree 2. In a list's Prüfer number, all the symbols are distinct, and a list on n nodes has $\frac{1}{6}n(n-1)(n+1)-n+1$ neighbors. Stars and lists have the smallest and largest phenotypic neighborhoods, respectively. All other spanning trees fall between these extremes, which Figure 2 plots as a function of the number n of nodes, and random trees are in general more similar to lists than to stars.

Figure 3 illustrates the low locality of most Prüfer numbers. For it, we examined the neighborhoods of

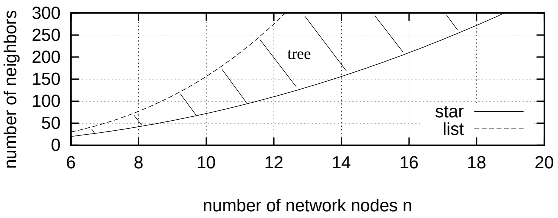


Figure 2: Phenotypic neighborhood sizes for lists and stars, as functions of the number of nodes. The values for all other trees lie between these curves.

1 000 spanning trees—stars, arbitrary trees, and lists—on $n = 16$ nodes. Figure 3(a) shows distributions of genotypic distances for neighboring spanning trees; that is, for spanning trees that differ in one edge. Figure 3(b) shows distributions of phenotypic distances for neighboring Prüfer numbers; that is, for Prüfer numbers that differ in one digit. Only for stars and trees similar to stars is the locality of the Prüfer coding high. In general, the phenotypes of genotypic neighbors are very different, and conversely.

2.3 Heritability

A coding has high heritability, with respect to a crossover operator, if offspring phenotypes consist mostly of substructures of their parents' phenotypes. When genotypes encode spanning trees, offspring trees should consist mostly or entirely of parental edges. Usually, with conventional operators, heritability will be low where locality is low. In Prüfer numbers, the meanings of genotypic symbols depend on their contexts, and k-point crossover will not preserve parental substructures in offspring phenotypes. Thus Prüfer-coded EAs using such a crossover will search effectively only near stars, where locality is high.

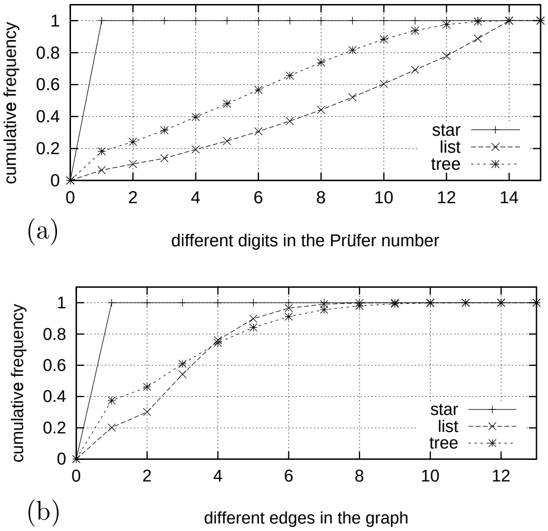


Figure 3: Distributions of (a) genotypic distances for neighboring spanning trees, and (b) phenotypic distances for neighboring Prüfer numbers, on 16 nodes.

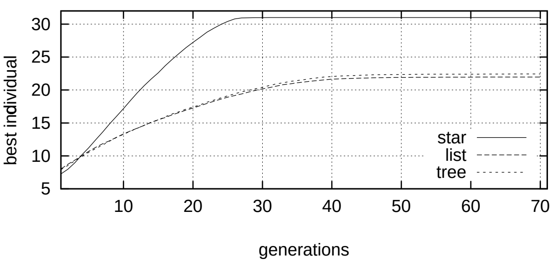


Figure 4: Performance of a Prüfer-coded EA on One-Max-Tree problems of 32 nodes, with optimum trees of various structures.

Experiments with a simple problem confirm that this is so. In the One-Max problem, a bit string's fitness is the number of 1s in it. In the One-Max-Tree problem (Rothlauf et al., 2000), an optimum spanning tree is specified, and the fitness of any tree is the number of edges that it shares with this target. An EA whose coding has sufficient heritability should solve this problem easily, by preserving edges of the target tree from parents to offspring.

A generational EA for One-Max-Tree encoded spanning trees as Prüfer numbers. It applied no mutation, only one-point crossover, and used $(\mu + \lambda)$ selection, with $\mu = \lambda = 1500$. Figure 4 illustrates the algorithm's performance on several One-Max-Tree instances with $n = 32$ nodes, whose target trees were variously stars, random trees, and lists. Only when the target tree's locality was high—*i.e.*, it was a star—did the EA find it. In other cases, the EA's search failed, and failed badly.

2.4 Where Locality is High

An EA will search effectively in regions of high locality, but when Prüfer numbers encode spanning trees, these regions are tiny. We extend the definition of neighbors to include trees whose Prüfer numbers differ in at most $i_{\max}$ digits ($i_{\max} \ll n$). The number of neighbors of stars is then $\sum_{i=0}^{i_{\max}} \binom{n-2}{i}(n-1)^i$; this value is $O(n^{2i_{\max}})$. The number of spanning trees on n vertices, and thus the size of the search space, is n^{n-2}. As Figure 5 illustrates, the proportion of these high-locality trees is small even for moderate n and diminishes exponentially as n grows. On problems of interesting size, Prüfer-coded EAs cannot succeed.

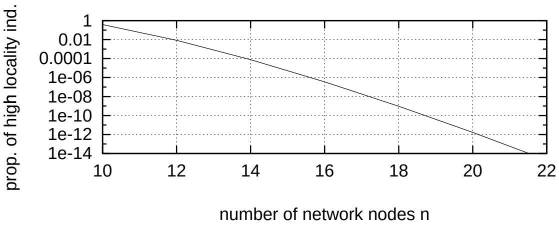


Figure 5: The proportion of spanning trees on n nodes whose Prüfer numbers have high locality, defined as differing from the Prüfer number of a star in no more than $i_{\max} = 5$ digits.

The following four sections compare codings of spanning trees in EAs for four substantive problems. They confirm the unsuitability of the Prüfer coding.

3 The Degree-Constrained Minimum Spanning Tree Problem

Given a weighted undirected graph G, the degree-constrained minimum spanning tree problem (d-MSTP) seeks a spanning tree on G of minimum weight whose degree does not exceed $d > 1$. This problem is NP-hard (Garey and Johnson, 1979, p. 206). Figure 6 shows an unconstrained spanning tree (of degree 5) and a spanning tree with maximum degree $d = 3$ on $n = 11$ points in the plane.

The degree constraint suggests that Prüfer numbers might perform well in an EA for this problem, since the degree of each node in a spanning tree is one more than the number of times its label appears in the tree's Prüfer number. Thus, it is easy to identify and repair Prüfer numbers whose trees violate the constraint.

Zhou and Gen (1997) presented an EA for the d-MSTP that encodes spanning trees as Prüfer numbers. However, it was tested only on very small, random problem instances. These were shown to be easily solved to optimality by greedy heuristics, branch-and-bound, or EAs using other encodings (Knowles and Corne, 2000; Krishnamoorthy et al., 1999; Raidl, 2000).

Other researchers have used other codings. Knowles and Corne (2000) described an EA in which genotypes are strings of integers that influence the order in which a variation of Prim's minimum spanning tree algorithm connects nodes to the spanning tree.

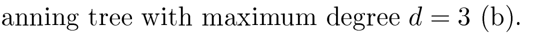


Figure 6: An unconstrained spanning tree (a) and a spanning tree with maximum degree $d = 3$ (b).

Palmer and Kershenbaum (1994) encoded spanning trees as strings of real-valued weights. The tree such a genotype represents is found by temporarily adding each node's weight to all the distances in which it participates, then applying Prim's algorithm to the modified distances. Raidl and Julstrom (2000) adapted this coding to the d-MSTP, using normally distributed, multiplicative weights.

Krishnamoorthy et al. (1999) compared two Prüfer-coded EAs, which differed in their crossover operators and breeding schemes, to a weight-coded EA and other optimization techniques. The Prüfer-coded EAs failed on all instances except the simplest; on average, the weight-coded EA performed best.

Recently, Raidl (2000) described an EA for the d-MSTP that stores the edges of candidate spanning trees directly in lists. The recombination operator uses parental edges to build a new spanning tree; to avoid violating the degree constraint, it must occasionally introduce edges that appear in neither parent. Mutation inserts a new edge and then removes an edge from the cycle so created.

Table 1 shows some characteristic results from Raidl (2000) on six hard, misleading 5-MSTP instances due to Knowles and Corne (2000). The table presents results for an EA that encoded spanning trees as Prüfer numbers (PR) and for three other algorithms: the EA of Knowles and Corne (K&C), the weight-coded EA of Raidl and Julstrom (2000) (Wts), and the edge-list EA of Raidl (2000) (LoE). Aside from its coding and operators, the Prüfer-coded algorithm was identical to the edge-list EA. Each algorithm was run 20 independent times on each instance. As in Krishnamoorthy et al. (1999), the Prüfer-coded EA performed worst, and its performance deteriorated most quickly as the problem size increased.

Table 1: Average weights of the best spanning trees found on 20 trials with four codings on six hard 5-MSTP instances.

5-MSTP	n	PR	K&C	Wts	LoE
`m050n1`	50	13.0	8.5	6.7	6.6
`m050n2`	50	14.1	7.8	6.0	5.8
`m100n1`	100	35.8	13.7	11.4	11.1
`m100n2`	100	39.2	15.5	11.9	11.4
`m200n1`	200	80.5	20.9	18.8	18.4
`m200n2`	200	87.3	26.4	20.3	19.5

4 The Optimal Communication Spanning Tree Problem

Consider a collection of nodes for which the communication demand between each pair of nodes is given. In the Optimal Communication Spanning Tree Problem (OCSTP), we seek a tree-structured network of minimum total cost that connects all the nodes. A link's flow is the sum of the communication demands between all pairs of nodes communicating directly or indirectly over the link. The cost for each link is not fixed *a priori* but depends on the length and capacity of the link. A link's capacity must satisfy the link's flow and this flow depends on the entire tree structure. Like other constrained spanning tree problems, the OCSTP is NP-hard (Garey and Johnson, 1979, p. 207). Figure 7 shows a communication spanning tree on 15 nodes and emphasizes the path connecting nodes 3 and 14.

Several researchers have encoded candidate communication trees as Prüfer numbers in EAs for this and related problems. For example, Kim and Gen (1999) considered problems in which the nodes were partitioned into users and service centers. Each user connected to exactly one service center, and Prüfer numbers represented spanning trees on the service centers.

Rothlauf et al. (2000) adapted NetKeys (Bean, 1992) to represent communication trees. In this representation, a genotype is a sequence of real-valued priorities associated with the edges that might appear in a communication tree. The decoding algorithm builds the represented tree by including edges in order of these priorities, skipping edges that form cycles. NetKeys support conventional evolutionary operators.

Prüfer numbers and NetKeys were compared in a generational EA on four instances of the OCSTP derived from a real-world problem whose nodes represent locations throughout Germany. Depending on the flow, each link is assigned one of four line types; these have distinct costs per unit length. In two of the instances (Type 1), each link's cost includes a fixed installation cost that does not depend on the link's length. For these instances, optimal solutions resemble stars. In the other two instances (Type 2), each link's cost depends only on its length and capacity. Here, optimal solutions are similar to simple minimum spanning trees.

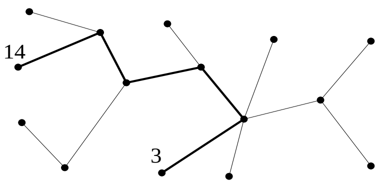


Figure 7: A communication spanning tree on 15 nodes, with the path connecting nodes 3 and 14 emphasized.

The EA in which the two codings were compared was that of Rothlauf et al. (2000). Its population size was 2 000 on all the problem instances. It selected parents in tournaments of size three and generated all offspring with uniform crossover; mutation was not used. The algorithm was run 50 independent times through 100 generations with each coding on each instance.

Table 2 summarizes the results of these trials. It presents the average percentages by which the costs of the best trees in each trial exceeded the costs of the single best tree identified on each instance. The algorithm's performance with Prüfer numbers was inferior to its performance with NetKeys and deteriorated more quickly with increasing problem size.

Table 2: The average percentage by which the best trees' costs with each coding exceeded the cost of the single best tree found, on each OCSTP instance.

OCSTP	size	Prüfer	NetKey
Type 1	16	3.38	0.02
	32	9.58	0.92
Type 2	16	5.71	0.78
	32	14.31	2.07

5 The Rectilinear Steiner Problem

Given a collection of points in the plane, a rectilinear Steiner tree (RStT) is a tree of horizontal and vertical line segments that connects them all. A RStT's length is the sum of its segments' lengths, with overlapping segments included only once. The search for a RStT of minimum length on a set of points is the rectilinear Steiner problem (RStP), and it is NP-hard (Garey and Johnson, 1979, p. 209).

In a rectilinear Steiner tree, the additional points where the segments meet are called Steiner points. In searching for a minimal RStT, we need consider as Steiner points only the unoccupied corners of the rectangles, with sides parallel to the axes, that each pair of points defines (Hanan, 1966). Each pair of points also defines an edge that may appear in a simple spanning tree. Assigning a Steiner point to each of the $n-1$ edges in a spanning tree specifies a RStT. Figure 8 shows a RStT and its underlying spanning tree.

Augmenting a Prüfer number with a string of $n-1$ binary symbols that indicate Steiner points encodes a rectilinear Steiner tree. Prüfer numbers and binary strings support conventional evolutionary operators.

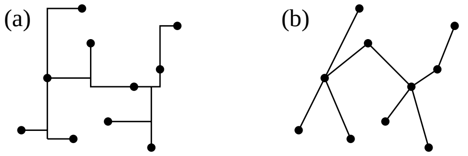


Figure 8: A rectilinear Steiner tree (a) and its underlying spanning tree (b).

An alternate coding augments Palmer and Kershenbaum's (1994) strings of weights, mentioned in Section 3, with binary strings. The binary symbols specify Steiner points for the edges of the spanning tree the weights represent. This coding also supports conventional operators; mutation changes parental weights and flips Steiner points.

A third coding of rectilinear Steiner trees is as lists of spanning tree edges augmented by Steiner point choices: the entry (17,25,1) represents the pair of rectilinear edges joining points 17 and 25 via, say, their right Steiner point. A list of $n-1$ such entries encodes a RStT. Crossover and mutation operators are based on union-find partitions of the given points.

Julstrom (2001) compared these codings in a generational EA with population size $2n$ and selection from tournaments of size four. With each coding, the algorithm was run 50 times on six RStP instances of from 50 to 150 points. Table 3 summarizes these tests; it lists the average length of the best trees found in each set of trials. The EA's performance is poor with the Prüfer coding, far better with augmented strings of weights, and best with lists of edges.

Table 3: Average lengths of the shortest RStTs the EA found with Prüfer numbers, augmented weights, and lists of edges.

RStP	n	Prüfer numbers	Augmented weights	Lists of edges
`eil50`	50	629	461	436
`st70`	70	1287	730	695
`eil75`	75	823	574	554
`rand80`	80	1566	835	780
`rand100`	100	1920	890	824
`rand150`	150	5472	2168	2031

6 The Fixed-Charge Transportation Problem

Consider distributing a commodity from m sources (factories or warehouses) to each of n destinations (consumers). The amounts of the commodity available at each source and required at each destination are known, and any source can ship to any destination. We seek a pattern of shipments, called a transportation plan, that minimizes the total cost of the deliveries. Figure 9(a) shows an instance of this problem, with $m = 3$ sources and $n = 2$ destinations. Figure 9(b) shows a transportation plan for it.

In the linear transportation problem, the cost of each link between a source and a destination depends linearly on the amount shipped; this problem is solvable in polynomial time (Edmonds and Karp, 1972). In the fixed-charge transportation problem (FCTP), each link also has a fixed cost that is invoked if that link is used; this problem is NP-hard (Guisewite and Pardalos, 1990).

A transportation tree consists of the links in a transportation plan and possibly additional links with capacity zero, as Fig. 9(c) illustrates. In such a tree, the amounts assigned to its edges are calculated by traversing the edges, iteratively selecting an edge incident to a leaf, and then assigning the maximum feasible amount to each. Transportation trees represent a subset of the feasible transportation plans, and that subset always contains at least one global optimum, so it is reasonable to restrict search to trees on the sources and destinations. Not all trees represent valid transportation plans, as Fig. 9(d) shows.

Li et al. (1998) described an EA for the FCTP that encodes transportation trees as Prüfer numbers. Gottlieb and Eckert (2000) extended this coding with repair mechanisms to ensure that each Prüfer number decodes to a feasible solution.

Vignaux and Michalewicz (1991) encoded transportation plans as permutations of the mn links between sources and destinations. A decoder scans the links in permutation order and assigns to each the largest amount of the commodity consistent with previous assignments and the problem's constraints. Gottlieb and Paulmann (1998) used this coding in an EA for the FCTP.

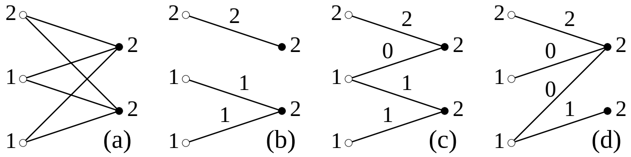


Figure 9: (a) A transportation problem with $m = 3$ sources (circles) and $n = 2$ destinations (dots). (b) A feasible transportation plan for the problem. (c) A transportation tree corresponding to the plan of (b). (d) A transportation tree whose plan violates the problem's constraints.

A list of its positive edges, each with the amount of the commodity assigned to it, can also represent a transportation tree. For this coding, there are two mutation operators. One introduces a new random edge and adjusts the tree accordingly; the other rearranges the edges incident to a randomly selected node. Crossover inserts half the edges of one parent into the other parent and removes existing edges as necessary.

These codings were compared in a steady-state EA for the FCTP. Its population contained 100 individuals (200 for `n3700` and `n370e`); it selected parents in tournaments of size two; it generated all offspring by crossover, then mutation; it always replaced the current worst genotype; and it did not allow phenotypic duplicates. The EA was run twelve independent times with each coding on six FCTP instances ranging in size from $m = 8$ and $n = 12$ to $m = 50$ and $n = 100$.

Table 4 summarizes these trials, some of which were reported in (Gottlieb and Eckert, 2000). For each instance and each coding, the table presents the percentage by which the average cost of the trials' best plans exceeded the cost of the single best plan found. Again, the EA performed worst with the Prüfer coding. Its performance was better with permutations and best when lists of edges represented transportation plans.

Table 4: The average percentage by which the twelve best plans' costs, with each coding, exceeded the costs of the best plans found, on six FCTP instances.

FCTP	Prüfer numbers	Permutations	Lists of edges
`bal8x12`	0.00	0.00	0.00
`ran4x64`	21.34	0.54	0.21
`ran14x18`	14.07	4.19	1.60
`ran16x16`	10.58	2.61	0.63
`n3700(50x100)`	65.96	13.79	0.83
`n370e(50x100)`	70.44	18.67	0.97

7 Conclusion

Prüfer numbers encode spanning trees on n nodes as strings of $n-2$ node labels. Several researchers have claimed that Prüfer numbers allow effective evolutionary search of spaces of spanning trees. We have demonstrated that this is not so.

Though Prüfer numbers support conventional evolutionary operators like k-point crossover and position-by-position mutation, only a negligible fraction of the genotype space of Prüfer numbers provides high locality and heritability under those operators. In general, mutated Prüfer numbers do not represent trees similar to those of their parents, nor do the offspring of crossover encode trees that consist mainly of substructures of the parental trees.

Empirical investigations on four NP-hard problems involving spanning trees confirm that the Prüfer coding is a poor choice in evolutionary algorithms. On every problem, an EA returned the worst results when Prüfer numbers encoded candidate trees, and much better results with other codings. Also, the performance of the Prüfer-coded versions deteriorated the most quickly as the problem instances got larger.

Why have some researchers reported good results with Prüfer numbers? In most cases, the problem instances were very small, so that any representation was adequate. In others, good solutions resembled stars, near which evolutionary search can be effective in spaces of Prüfer numbers.

Another disadvantage of Prüfer numbers is that they correspond to unconstrained spanning trees in complete graphs. When the underlying graph is not complete or the problem is otherwise constrained, an offspring Prüfer number will not in general represent a valid solution. Offspring can be repaired, but such operations further degrade the representation's locality and heritability. Operators that enforce locality and heritability on Prüfer numbers must decode them and operate on the decoded representations.

For three of our four problems, one of the codings of candidate solutions was lists of spanning tree edges, sometimes augmented with information such as Steiner points or commodity amounts. The EAs returned their best results with these codings. With appropriate operators, lists of edges show high locality and heritability, and they can be manipulated to satisfy constraints. This suggests that lists of edges may be a generally effective representation of spanning trees for evolutionary search.

References

F. N. Abuali, D. A. Schoenefeld, and R. L. Wainwright. Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, editors, *Proceedings of the 1994 ACM Symposium on Applied Computing*, pages 242–246. ACM Press, 1994.

J. C. Bean. Genetics and random keys for sequencing and optimization. Technical Report 92-43, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1992.

A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889.

J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

S. Even. *Algorithmic Combinatorics*. The Macmillan Company, New York, 1973.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

M. L. Gargano, W. Edelson, and O. Koval. A genetic algorithm with feasible search space for minimal spanning trees with time-dependent edge costs. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, page 495. Morgan Kaufmann, 1998.

M. Gen and Y. Li. Spanning tree-based genetic algorithms for the bicriteria fixed charge transportation problem. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala, and W. Porto, editors, *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pages 2265–2271. IEEE Press, 1999.

J. Gottlieb and C. Eckert. A comparison of two representations for the fixed charge transportation problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 345–354. Springer, 2000.

J. Gottlieb and L. Paulmann. Genetic algorithms for the fixed charge transportation problem. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 330–335. IEEE Press, 1998.

G. M. Guisewite and P. M. Pardalos. Minimum concave-cost network flow problems: Applications, complexity, and algorithms. *Annals of Operations Research*, 25:75–100, 1990.

M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, 14 (2):255–265, 1966.

B. A. Julstrom. Encoding rectilinear Steiner trees as lists of edges. In G. Lamont, J. Carroll, H. Haddad, D. Morton, G. Papadopoulos, R. Sincovec, and A. Yfantis, editors, *Proceedings of the 2001 ACM Symposium on Applied Computing*, pages 356–360. ACM Press, 2001.

J. R. Kim and M. Gen. Genetic algorithm for solving bicriteria network topology design problem. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala, and W. Porto, editors, *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pages 2272–2279. IEEE Press, 1999.

J. Knowles and D. Corne. A new evolutionary approach to the degree constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4(2):125–134, 2000.

M. Krishnamoorthy, A. T. Ernst, and Y. M. Sharaiha. Comparison of algorithms for the degree constrained minimum spanning tree. Technical report, CSIRO Mathematical and Information Sciences, Clayton, Australia, 1999.

Y. Li, M. Gen, and K. Ida. Fixed charge transportation problem by spanning tree-based genetic algorithm. *Beijing Mathematics*, 4(2):239–249, 1998.

C. C. Palmer and A. Kershenbaum. Representing trees in genetic algorithms. In D. Schaffer, H.-P. Schwefel, and D. B. Fogel, editors, *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 379–384. IEEE Press, 1994.

H. Prüfer. Neuer Beweis eines Satzes ueber Permutationen. *Archiv für Mathematik und Physik*, 27: 742–744, 1918.

G. R. Raidl. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In C. Fonseca, J.-H. Kim, and A. Smith, editors, *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pages 104–111. IEEE Press, 2000.

G. R. Raidl and B. A. Julstrom. A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 440–445. ACM Press, 2000.

F. Rothlauf and D. E. Goldberg. Pruefernumbers and genetic algorithms: A lesson on how the low locality of an encoding can harm the performance of GAs. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 395–404. Springer, 2000.

F. Rothlauf, D. E. Goldberg, and A. Heinzl. Network random keys – a tree network representation scheme for genetic and evolutionary algorithms. Technical Report No. 8/2000, University of Bayreuth, Germany, 2000.

G. A. Vignaux and Z. Michalewicz. A genetic algorithm for the linear transportation problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 21 (2):445–452, 1991.

G. Zhou and M. Gen. Approach to degree-constrained minimum spanning tree problem using genetic algorithm. *Engineering Design & Automation*, 3(2): 157–165, 1997.

Solving Fixed Channel Assignment Problems by An Evolutionary Approach

JORNG-TZONG HORNG
Department of Computer Science
and Information Engineering
National Central University,
Jungli, 320, Taiwan

Ming-Hui Jin
Department of Computer Science
and Information Engineering
National Central University,
Jungli, 320, Taiwan

CHENG-YAN KAO
Department of Computer Science
and Information Engineering,
National Taiwan University,
Taiwan

Abstract

Increasing the channel reusability is necessary for reducing the call-blocking rate in any cellular networks with limited bandwidth resources and large-scale subscribers. To increase the channel reusability, we need an efficient channel assignment algorithm. In this paper, we propose a genetic approach to solve fixed channel assignment problems. From the experimental results, we find our genetic approach performs well and our algorithm does find lower-cost assignments.

1 INTRODUCTION

A lack of available bandwidth resources is a heavy bottleneck for the capacity of personal communication services (PCS) networks. In addition, a large amount of PCS subscribers makes the limited frequency problem more emergent. The situation can be improved by many approaches. For example, the time division multiple access (TDMA) protocol is proposed to promote the usability of each available frequency. On the other hand, base stations are built to decrease the size of each cell so that the demands of bandwidth requirements for each cell could be reduced. Moreover, increase the reusability of existing frequencies by assigning appropriate frequency to appropriate cells is also an efficient scheme.

The channel assignment problems are proposed to maximize the channel reusability under a set of given constraints so that the capacity of the PCS subscribers could be increased. The channel assignment problem could be classified into two categories: 1) fixed channel assignment (FCA), where channels are fixed allocated to each cell and 2) dynamic channel assignment (DCA), where all channels which are available for every cell, are allocated dynamically upon request. Normally, DCA has better performance than FCA except under heavy traffic load condition, where FCA outperforms DCA [2, 4]. Since heavy traffic load is expected in the future generation of cellular radio networks, an efficient FCA scheme provides high spectrum usage efficiency is desired. How to assign all the available frequencies to each cell appropriately is an important issue for FCA.

Three types of electromagnetic compatibility (EMC) constraints need to be considered in the problem of channel assignments [1-3]. They are 1) the cochannel constraint (CCC), where the same channel cannot be assigned to certain pairs of radio cells simultaneously; 2) the adjacent channel constraint (ACC), where channels adjacent in frequency spectrum cannot be assigned to adjacent radio cells simultaneously; and 3) the cosite constraint (CSC), where channel assigned in the same radio cell must have a minimal separation in frequency between each other. Thus, cellular cooperation provides a matrix called the compatibility matrix to represent the EMC constraint for channel assignment. In the compatibility matrix, each element c_{ij} describes the minimum frequency separation between a frequency used simultaneously in cells i and j. For FCA, each feasible solution is required to satisfy an extra constraint called the demand constraint that states the minimal channel requirement of each cell. To simplify the channel assignment problem, the demand constraint is represented by a vector called the least demand vector in which each component d_i corresponds to the least amount of frequencies required in cell i is provided for channel assignment.

The channel assignment problem in cellular network is known to belong to the class of NP-complete problems [2, 15]. To achieve the optimal solution of fixed channel assignment problems, most proposed algorithms try to minimize the amount of necessary channels under satisfying a set of given constraints [1-9]. However, the total number of available frequencies are given and fixed in many situations. Minimizing the amount of necessary frequencies becomes meaningless for such applications. To find an appropriate channel assignment for the cellular networks with limited available channels, we proposed a cost model in [1] in which the available frequencies are given and fixed and the EMC constraints and demand constraint are relaxed. Our optimization problem tries to minimize the damages from violating the EMC constraints and demand constraint. To solve the optimization problem in [1], we proposed a genetic algorithm to solve the fixed channel assignment problems.

This paper is organized as follows. In Section 2, we introduce the cost model for fixed channel assignment problem with fixed amount of available frequencies. In Section 3, the genetic operators used to minimize the cost model are presented. Experiments and results are given in Section 4. The conclusion is drawn in Section 5.

2 FIXED CHANNEL ASSIGNMENT PROBLEMS

2.1 REQUIREMENTS OF CHANNEL ASSIGNMENT UNDER FIXED AMOUNT FREQUENCIES

The total amount of blocked calls and interference between calls are the two major considerations of channel assignment. Both blocking and interference brings certain degree of damages to the PCS. Thus, an ideal assignment algorithm should bring feasible solutions that guarantee low blocking rate and low degree of interferences.

Damages of Blocked Calls

At a busy base station call attempts that fail because there are no available channels are called blocked calls. It is clear that the more blocked calls bring the more damages of the PCS. Thus, an ideal channel assignment should guarantee that the amount of total blocked calls of all cells could be low enough. Let X_i be the random variable of required channels in cell i and H_i be the amount of assigned frequencies to cell i, then $\sum_{j=H_i+1}^{n_i} P(X_i = j)(j - H_i)$ is the expected amount of blocked calls in cell i and then we would like to minimize the cost $\sum_{i=1}^{N} \sum_{j=H_i+1}^{n_i} P(X_i = j)(j - H_i)$ where N is the number of all cell and n_i is the amount of mobiles in cell i.

Damages of Interference

If a mobile of cell i get a channel whose frequency is p and another mobile of cell j get a channel whose frequency is q, according to the definition of compatibility matrix, no interference between the two calls if $\|p - q\| \ge c_{ij}$. But if $\|p - q\| < c_{ij}$, the qualities of the two communications would be in negative proportion to the distance $\|p - q\|$. In this reason, an ideal channel assignment should satisfy the EMC constraint whenever the compatibility matrix is estimated.

But in many real cases, the amount of available frequencies is given and fixed. The feasible solutions that satisfy the EMC constraint may bring high blocking rate condition. High blocking rate not only reduce the quality of service but also reduce the service charge to its subscribers. Thus, the EMC constraint may need to be violated when the amount of available frequencies is insufficient.

Violating EMC constraint would bring some degree of disadvantage to the mobiles that are interfered. And the degree of disadvantage should be in proportion to the degree of interference that depends on the frequency distance (i.e., how many Hz between them) and the power it suffered. To construct a cost function for fixed channel assignment problem, we quantify the damages from channel interference as following. If we assign frequency p to cell i and q to cell j, then the degree of damage caused by interference from this assignment is defined to be *f(i, j, p, q)* as follows.

$$f(i,j,p,q)=\begin{cases}0 & \textit{if } |p-q|\ge c_{i,j}\\ f_{i,p}f_{j,q}\Psi_C(c_{i,j}-|p-q|) & \text{if } |\text{p - q}|<\text{c}_{\text{ij}} \text{ and i = j}\\ f_{i,p}f_{j,q}\Psi_A(c_{i,j}-|p-q|) & \text{otherwise.}\end{cases}$$

where

$$f_{i,p}=\begin{cases}1 & \textit{if the } p^{th} \textit{ frequency is assigned to the } i^{th} \textit{ cell.}\\ 0 & \textit{Otherwise.}\end{cases}$$

Ψ_C and Ψ_A are two strictly increasing functions.

2.2 ASSUMPTIONS AND DENOTATIONS

To formulate the cost function to measure the damage of each assignment, we first give and descirbe the following notations:

D: The least demand vector with N components in which each component d_i of vector D corresponds to the least amount of frequencies required in cell i.

C: The compatibility N×N matrix in which each element $c_{i,j}$ describe the minimum frequency separation between a frequency used simultaneously in cells i and j. This matrix is used to assure that the computed frequency assignment does not lead to any interference between different calls.

F: The frequency assignment matrix in which each element $f_{i,j}$ is defined in Subsection 2.1.

F_i: The amount of frequencies assigned to cell i.

D: A frequency provides D channels under TDMA approach.

n_i: The expected amount of mobiles stays in cell i.

$$X_{i,j} = \begin{cases} 1 & \text{if the } j^{th} \text{ mobile of cell } i \text{ wish to own a channel} \\ 0 & \text{otherwise} \end{cases}$$

$$X_i = \sum_{j=1}^{n_i} X_{i,j}$$

μ_i: The expected amount of requests channels (No matter the request is failed or success) in cell i.

σ_i: The standard deviation of the amount of requests channel (No matter the request is failed or success) in cell i.

In general, the load of each cell is maintained by system. μ_i and σ_i could be estimated from long-term history of load data of base station i. It is acceptable to assume that the n_i random variables $X_{i,j}$ are independent. Thus, we make the following assumptions:

1. The parameters μ_i and σ_i could be estimated from the communication load of cell i.
2. For each $1 \le i \le N$ and $1 \le j \ne k \le n_i$, the two random variables $X_{i,j}$ and $X_{i,k}$ are independent and $0 < E[X_{i,j}] < 1$.
3. For each $1 \le i \le N$, $\mu_i = \sum_{j=1}^{n_i} E[X_{i,j}]$ and $\sigma_i = \sqrt{\sum_{j=1}^{n_i} Var(X_{i,j})}$

With the above denotations and assumptions, we assume, for each $1 \le i \le N$, the random variable X_i is close to the normal random variable with parameters μ_i and σ_i by theorem 1 below.

Theorem 1: If, for each $1 \le j \ne k \le n_i$, the two random variables $X_{i,j}$ and $X_{i,k}$ are independent, then

$$P\left\{\frac{X_i - \mu_i}{\sigma_i} \le a\right\} \to \Phi(a) \qquad as\ n_i \to \infty$$

where $\mu_i = \sum_{j=1}^{n_i} E[X_{i,j}]$, $\sigma_i = \sqrt{\sum_{j=1}^{n_i} Var(X_{i,j})}$ and Φ is the distribution function of a standard normal random variable.

Proof:

It is clear that $X_{i,j}$ are all uniformly bounded. Because $0 < E[X_{i,j}] < 1$, $Var(X_{i,j}) = E[X_{i,j}]*(1-E[X_{i,j}]) \ne 0$ is clearly true and hence we have $\lim_{n_i \to \infty} \sum_{j=1}^{n_i} Var(X_{i,j}) = \infty$.

Because $X_{i,1}$, $X_{i,2}$, …, X_{i,n_i} forms a sequence of independent random variables, according to the central limit theorem for independent random variables [10, 11], we have

$$P\left\{\frac{\sum_{j=1}^{n_i}(X_{i,j} - E[X_{i,j}])}{\sqrt{\sum_{j=1}^{n_i} Var(X_{i,j})}} \le a\right\} \to \Phi(a) \qquad as\ n_i \to \infty$$

This completes our proof.

2.3 PROBLEM FORMULATION

In subsection 2.2, we make an assumption that the two parameters μ_i and σ_i could be estimated from the load data of cell i. In this subsection we would like to construct the cost function whose parameters are μ_i and σ_i only.

Let's assume that each frequency provides D channels, whenever all $f_{i,p}$ have been decided, the total damages due to the assignment is $\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{Z}\sum_{q=1}^{Z} f(i,j,p,q)$ + $\alpha \sum_{i=1}^{N} \sum_{j=DF_i+1}^{n_i} P(X_i = j)(j - DF_i)$ where α is the relative weight of damages from blocked calls to the damages from call interference. To calculate the cost for each assignment, we could use $\frac{1}{\sqrt{2\pi\sigma_i}} \int_{DF_i}^{\infty} (y - DF_i) e^{-\frac{1}{2}(\frac{y-\mu_i}{\sigma_i})^2} dy$ to be an approximation of $\sum_{j=DF_i+1}^{n_i} P(X_i = j)(j - DF_i)$ because X_i is close to the normal random variable with parameter μ_i and σ_i for all i. Thus, we could state the fixed channel assignment problem as follows:

$$\text{Min} \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{Z}\sum_{q=1}^{Z} f(i,j,p,q) + \alpha \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma_i}} \int_{DF_i}^{\infty} (y - DF_i) e^{-\frac{1}{2}(\frac{y-\mu_i}{\sigma_i})^2} dy \qquad (1)$$

Subject to

$$f_{i,p} = 0 \text{ or } 1 \qquad for\ 1 \le i \le N\ and\ 1 \le p \le Z \qquad (2)$$

where

$$F_i = \sum_{p=1}^{Z} f_{i,p} \qquad for\ 1 \le i \le N. \tag{3}$$

$$f(i,j,p,q) = \begin{cases} 0 & if\ |p-q| \ge c_{i,j} \\ f_{i,p} f_{j,q} \Psi_C (c_{i,j} - |p-q|) & if\ |p-q| < c_{ij}\ and\ i = j \\ f_{i,p} f_{j,q} \Psi_A (c_{i,j} - |p-q|) & otherwise. \end{cases} \tag{4}$$

+c_{ij}, Z and N are the same as notations in Section 2.2.

3 OUR EVOLUTIONARY APPROACH

Genetic algorithms have been applied to various optimization problems. In general, they use a penalty function to encode constraints and allow a search for undesirable solution, e.g., a solution may exist many high degree interferences and high call blocking rates. Allowing a search for illegal nodes may prevent falling down into a local minimum and generate a better solution. It is well suited for the cost optimization problem from (1) – (4). In this section, we first review the fundamentals of genetic algorithms and then describe the related operators and functions for the fixed channel assignment problem.

3.1 PRINCIPLES OF GENETIC ALGORITHMS

Genetic algorithm search methods are based on evolution and natural genetics. Fundamental to the GA structure is the encoding mechanism for representing the optimization problem's variables. In each case, the encoding mechanism maps each solution to a unique binary string. Only the fittest individuals survive and reproduce, a natural phenomenon called *the survival of the fitness* [13].

The Genetic algorithm has two steps, a reproduction step and a recombination step. The recombination process, consisting of crossover and mutation operators, is based on natural heredity to generate offspring in the next generation. The reproduction step, a selection mechanism, is used to eliminate the worse individuals. Restated, *who shall live and who shall die* must be determined [12]. From another perspective, GA is the balanced combination of an exploration of new regions in the search space and an exploitation of previously sampled regions. The crossover operator frequently performs a larger jump in the search space. Except for escaping from local optima, these jumps are also an exploration skill. Although performing fewer changes in the solution operations, the mutation operator performs more local search work. This task involves basic local search technique and other exploitation skills [13].

Genetic algorithms can produce an adequate solution from a large search area by accumulating candidate solutions to a problem. The GA is a general method for solving *search for solutions* and *search for paths to goal problems*. Diverse areas such as optimization, machine learning, immune systems, and NP-complete or NP-hard problem have profited from these methods [14]. The fixed channel assignment problem in (1) – (4) is appropriate for solutions using GAs owing to its' *search for solutions, optimization,* and *NP-hard* properties.

The nature of GAs that directs the evolution of a program to the optimal solution creates the greedy phenomenon. This phenomenon easily traps into local optimal. To avoid this problem, heuristics are usually used to aid in the escape from local optimal area. Restated, combining GAs with heuristic ideas can lead to the evolutionary convergence to global optima. Moreover, applying the above concept allows us to utilize a novel concept as mutation guidance to obtain a better solution.

3.2 CHROMOSOMES AND INITIAL POPULATION

The chromosome for FCAP is represented by an N×Z matrix as Fig. 1 below.

← Z columns →

↑ N rows ↓

$f_{1,1}$	$f_{1,2}$	…	$f_{1,p}$	…	$f_{1,Z}$
$f_{2,1}$	$f_{2,2}$	…	$f_{2,p}$	…	$f_{2,Z}$
…					
$f_{i,1}$	$f_{i,2}$	…	$f_{i,p}$	…	$f_{i,Z}$
$f_{N,1}$	$f_{N,2}$	…	$f_{N,p}$	…	$f_{N,Z}$

Fig.1. Chromosome of FCAP

According to constraint (2), $f_{i,p} \in \{0, 1\}$ for all $1 \le i \le N$ and $1 \le p \le Z$. To fasten the speed of convergence, generating the first generation by using some heuristics is another attemptable approach that may bring precocity of the evolution process. We design a heuristic below to generate initial population each solution of which has low interference inside each cell.

Initial Heuristic for FCAP

```
For(i = 0; i < N; i++)
{
    Randomly decide an integer Fi.
    Randomly choose an integer sequence $1 \le d_1 < d_2 < \ldots < d_{Fi} \le Z$ such that $d_{i+1} - d_i$ is around Z/Fi.
    Set $f_{ij} = 1$ for all $j \in \{d_1, d_2, \ldots, d_F\}$
}
```

3.3 GENETIC OPERATORS

Crossover

Given two parents $A = (a_{ij})_{N \times Z}$ and $B = (b_{ij})_{N \times Z}$. If $a_{ik} = b_{ik}$ for some i and k, then the assignment decision (whether the k^{th} frequency should be assigned to the i^{th} cell) is agreed by both A and B and hence we should be inherited by their children. If $a_{ik} \neq b_{ik}$, then we choose one of them randomly to be their children's decision. Based on this concept we define a crossover operator ⊕ as follows:

For any two chromosome $A = (a_{ij})_{N \times Z}$ and $B = (a_{ij})_{N \times Z}$, $A \oplus B = (c_{ij})_{N \times Z}$ where

- $P(c_{ij} = a_{ij}) = \dfrac{Cost(A)}{Cost(A) + Cost(B)}$ and

  $P(c_{ij} = b_{ij}) = \dfrac{Cost(B)}{Cost(A) + Cost(B)}$
- Cost(A) is calculated by cost function (1)

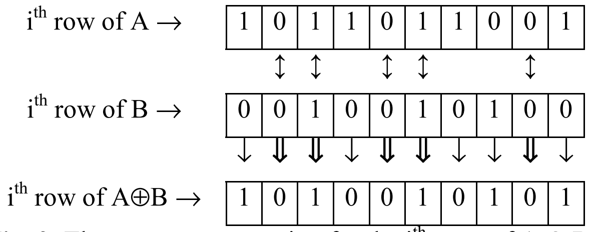


Fig. 2. The crossover operation for the i^{th} rows of A ⊕ B

Local Search

Local search is a unary operator that is used to find a local optimal around each chromosome. In this paper we design two local search operators for reducing the intra-cell and inter-cell interferences, respectively. To reduce the intra-cell interference, we design a heuristic called position adjustment heuristic (PAH) to adjust the distance between channels inside each cell.

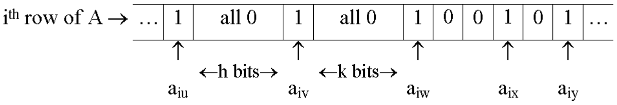


Fig. 3. The checking region and degree of freedom

Fig. 3 shows an example for describing the idea of PAH where $C_{ii} > 2$ and both h and k are greater than C_{ii}. In this situation, a_{iv} could move 1 to h-C_{ii} bits leftward and move 1 to k-C_{ii} bits rightward freely without interfering to both a_{iu} and a_{iw} but a_{iw} could only move 1 to k-C_{ii} bits leftward freely without interfering to any other frequency in the same cell. It is clear that we should first move a_{iw} leftward in this example. Based on this concept, we formally define the freedom of each bit a_{ij} in each chromosome to be (h_{ij}, k_{ij}) where

- ***$h_{ij} = k_{ij} = -\infty$ if $a_{ij} = 0$. If $a \neq 0$, then***
  - ***$h_{ij} = \min\{j-t > 0 | a_{it} = 1\} - Cii$ if $\{a_{it} | t < j$ and $a_{it} = 1\} \neq \phi$.***
  - ***$h_{ij} = j-1$ if $\{a_{it} | t < j$ and $a_{it} = 1\} = \phi$.***
  - ***$k_{ij} = \min\{t-j > 0 | a_{it} = 1\} - Cii$ if $\{a_{it} | j < t$ and $a_{it} = 1\} \neq \phi$.***
  - ***$h_{ij} = Z-j-1$ if $\{a_{it} | j < t$ and $a_{it} = 1\} = \phi$.***

According to the definition above, if the freedom of a_{ij} is (h_{ij}, k_{ij}), then we could move a_{ij} at most h_{ij} bits leftward and move a_{ij} at most k_{ij} bits rightward from its original position. Based on the above discussion, we design our local search operation for reducing the intra-cell interference as follow

Local-Search operator (APH)

```
For(i = 0; i < N; i++)      {
    Calculate the freedom of $a_{ij}$ for all j
    Let M = { j | $-\infty < h_{ij}k_{ij} < 0$};
    While(There exist some j such that $-\infty < h_{ij}k_{ij} < 0$) {
      Find $j \in M$ such that $\min(h_{ij}, k_{ij}) \leq \min(h_{it}, k_{it})$ for all $t \in M$;
      If($h_{ij} < 0$)
         Move x bits rightward for some $1 \leq x \leq k_{ij}$
      Else
         Move x bits leftward for some $1 \leq x \leq h_{ij}$
      Calculate the freedom of $a_{ij}$ for all j
      Let M = { j | $-\infty < h_{ij}k_{ij} < 0$};
    }
}
```

To reduce the inter-cell interference, we design a heuristic called row-rotation heuristic (RRH) to adjust the frequency interference between adjacent cells. To facilitate the statement of our mutation operation, we define a function $\Re$ called row-rotate operation as below.

For each chromosome $A = (a_{ij})_{N \times Z}$, $1 \leq x \leq N$ and $0 \leq y \leq Z-1$,
$\Re(A, x, y) = (r_{ij})_{N \times Z}$
where
$r_{ij} = a_{ij}$ if $i \neq x$
$r_{xj} = a_{xw}$ satisfying $w \equiv j-y$ mode Z

Fig. 4 below shows an example of Row-Rotate operation for y = 3 and we state our mutation operation below.

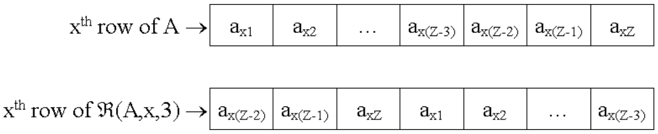


Fig. 4. The Row-Rotate operation with y = 3

Local-Search operator (RRH)

```
For(i = 0; i < N; i++)  {
    Find an integer R satisfying Cost($\Re$(A, i, R)) $\leq$ Cost($\Re$(A, i, r)) for all integer r.
        A = $\Re$(A, i, R)
}
```

4 EXPERIMENTS AND RESULTS

The experiments were performed on a Pentium II-400 PC with 125 MB memory running windows 2000 operating system. For all the experiments, the population size of each problem is equal to the number of available frequency. The maximal generation of each experiment is 1000 and evolution stops whenever the fitness of 10 contiguous generations are not changed.

4.1 BENCHMARK PROBLEMS

Eight benchmark problems are proposed and examined in our experiments. The compatibility matrices C_3 – C_5 for the eight problems are given in [9] and we list them in Appendix. Fig. 5 shows the three communication load tables and each one of them contains the mean and standard derivation of the communication loads for each cell. Table 1 shows the problem specifications for the five problems. In all of our experiments, we adopt the functions $\Psi_C(x) = 5^{x-1}$ and $\Psi_A(x) = 5^{2x-1}$ to measure the cost of interference and the weight of blocking cost $\alpha = 10000$.

4.2 EXPERIMENTAL RESULTS

Table 2 compares the best and average cost of simple GA [1] and our designed genetic operators. In simple GA, the initial populations for their evolutionary process are randomly generated without any heuristic. And we adopt the heuristic in Subsection 3.2 to generate our initial population. According to Table 2, we conclude that our designed algorithm does improve the quality of solutions in all of the 8 problems.

5 CONCLUSION

In this paper we introduce a new cost model for fixed channel assignment problems with limited bandwidth constraint. In the new cost model, the EMC constraints are relaxed by converting them into cost of damages. The new cost model is concluded from prudent statistical analogy in Section 2. To find acceptable assignments we design several genetic operators for finding low-cost solutions. The experimental results show our algorithm does helpful in finding low-cost assignments.

From table 2 in Section 4, we observe that the average generations of our genetic approach are still small. It shows that the convergent speed of the evolution processes is too fast. Convergent too fast to make individuals precocious does not a good phenomenon for evolutionary process. It in general makes solutions lapse into local optimal [12]. This may because the operators are over heuristic. To improve the precoious phenomenon, we'll try to design better operators such as mutations that could generate children with higher variation.

Cell	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
μ	5	5	5	8	12	25	30	25	30	40	40	45	20	30	25	15	15	30	20	20	25
σ	1.2	1.1	1.15	1.6	2.24	5	5.72	5	6.1	8.1	8.02	9.11	4.07	5.99	5.61	3.14	3.07	5.86	4.12	3.98	5.18

(a)

Cell	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
μ	8	25	8	8	8	15	18	52	77	28	13	15	31	15	36	57	28	8	10	13	8
σ	1.61	4.88	1.52	1.49	1.61	3.11	3.52	9.73	11.62	5.1	2.15	2.66	4.72	2.77	4.93	8.64	3.92	1.25	1.72	2.14	1.27

(b)

Cell	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
μ	59.38	105.56	95.79	92.83	61.35	191.81	39.53	62.84	42.58	34.73	36.02	41.59	80.7	35.67	61.91	42.22	65.61	52.45	85.78	33.9	80.62
σ	6.23	8.98	14.07	11.08	5.61	21.62	6.78	8.72	10.3	6.63	7.92	6.29	11.34	6.02	5.61	5.96	7.39	5.86	9.92	3.52	6.18

(c)

Fig. 5. Communication load tables for experimental problems. (a) Table D_1. (b) Table D_2. (c) Table D_3

Table 1. Problem specifications

Problem	No. of Cells (N)	No. of Available Frequencies (Z)	Compatibility Matrix (C)	Communication Load Table (μ, σ)
1	21	60	C_3	D_3
2	21	60	C_4	D_1
3	21	60	C_5	D_1
4	21	60	C_4	D_2
5	21	60	C_5	D_2
6	21	40	C_5	D_1
7	21	40	C_5	D_2
8	21	64	C_5	D_3

Table 2. Solutions of the problems

Problem	Simple GA in [1]		Our GA approach		
	Cost of Best known Solution	Average Cost Average Generation	Cost of Best known Solution	Average Cost Average Generation	Improve
1	217.947	325.714	203.366	302.556	6.7%
2	276.623	362.558	271.366	342.498	1.9%
3	2013.751	2954.417	1957.366	2864.122	2.8%
4	950.995	1025.628	906.2985	1002.442	4.7%
5	4495.609	4752.644	4302.298	4585.366	4.3%
6	4857.711	5111.629	4835.366	5076.223	0.46%
7	21700.624	22524.886	20854.3	21968.366	3.9%
8	58089.148	61098.558	53151.57	60715.442	8.5%

References

[1] M. H. Jin, H. K. Wu, J. Z. Horng, and C. H. Tsai, "An Evolutionary Approach to Fixed Channel Assignment Problems with Limited Bandwidth Constraint", will appear in the Proceeding of IEEE ICC 2001 .

[2] C. Y. Ngo and V. O. K. Li, "Fixed Channel Assignment in Cellular Radio Networks Using a Modified Genetic Algorithm," IEEE Transactions on Vehicular Technology, vol. 47, no. 1, pp. 163-171, 1998.

[3] K. N. Sivarajan, R. J. McEliece, and J. W. Ketchun, "Channel assignment in cellular radio," in Proc. 39th IEEE Vehicular Technology Conference, pp. 846-850, May 1989.

[4] P. Raymond, "Performance analysis of cellular networks," IEEE Transactions on Communications, vol. 39, no. 12, pp. 1787-1793, 1991.

[5] B. Dirk and K. Ulrich, "A New Strategy for the Application of Genetic Algorithms to the Channel-Assignment Problem," IEEE Transactions on Vehicular Technology, vol. 48, no. 4, 1999.

[6] W. K. Hale, "Frequency assignment: theory and applications," Proc. IEEE, vol. 68, no. 12, pp. 1497-1514, 1980.

[7] F. Box, "A heuristic technique for assignment frequencies to mobile radio nets," IEEE

Transactions on Vehicular Technology, vol. 27, pp. 57-64, 1977.

[8] S. Kim and S. L. Kim, “A two-phase algorithm for frequency assignment in cellular mobile systems,” IEEE Transactions on Vehicular Technology, vol. 43, no. 3, pp. 542-548, 1994.

[9] N. Funabiki and Y. Takefuji, “A neural network parallel algorithm for channel assignment problems in cellular radio networks,” IEEE Transactions on Vehicular Technology, vol. 41, no. 4, pp. 430-437, 1992.

[10] Edward R. Dougherty, Probability and Statistics For The Engineering, Computing And Physical Sciences, Prentice-Hall International, INC.

[11] Sheldon Ross, A First Course in Probability, Fifth edition, Prentice-Hall International, INC.

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

[13] M. Stinivas, and L. M. Patnaik, “Genetic algorithms: A survey”, *Computer Journal 27(2)*, pp. 17-26, 1994

[14] M. Mitchell, *An Introduction to Genetic Algorithms*. A Bradford Book, The MIT Press, Cambridge, Massachusetts, London, England, 1996.

[15] A. Gamst and W. Rave, “On frequency assignment in mobile automatic telephone systems,” in *Proc. IEEE GLOBECOM ’82*, pp. 309–315.

Appendix: The compatibility matrices $C_3 – C_5$

$$\begin{matrix}
5&1&1&0&0&1&1&1&1&0&0&0&0&1&1&1&0&0&0&0&0\\
1&5&1&1&0&0&1&1&1&1&0&0&0&0&1&1&1&0&0&0&0\\
1&1&5&1&1&0&0&1&1&1&1&0&0&0&0&1&1&1&0&0&0\\
0&1&1&5&1&0&0&0&1&1&1&1&0&0&0&0&1&1&0&0&0\\
0&0&1&1&5&0&0&0&0&1&1&1&0&0&0&0&0&1&0&0&0\\
1&0&0&0&0&5&1&1&0&0&0&0&1&1&1&0&0&0&0&0&0\\
1&1&0&0&0&1&5&1&1&0&0&0&1&1&1&1&0&0&1&0&0\\
1&1&1&0&0&1&1&5&1&1&0&0&0&1&1&1&1&0&1&1&0\\
1&1&1&1&0&0&1&1&5&1&1&0&0&0&1&1&1&1&1&1&1\\
0&1&1&1&1&0&0&1&1&5&1&1&0&0&0&1&1&1&0&1&1\\
0&0&1&1&1&0&0&0&1&1&5&1&0&0&0&0&1&1&0&0&1\\
0&0&0&1&1&0&0&0&0&1&1&5&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&5&1&1&0&0&0&0&0&0\\
1&0&0&0&0&1&1&1&0&0&0&0&1&5&1&1&0&0&1&0&0\\
1&1&0&0&0&1&1&1&1&0&0&0&1&1&5&1&1&0&1&1&0\\
1&1&1&0&0&0&1&1&1&1&0&0&0&1&1&5&1&1&1&1&1\\
0&1&1&1&0&0&0&1&1&1&1&0&0&0&1&1&5&1&1&1&1\\
0&0&1&1&1&0&0&0&1&1&1&1&0&0&0&1&1&5&0&1&1\\
0&0&0&0&0&0&1&1&1&0&0&0&0&1&1&1&1&0&5&1&1\\
0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&1&1&1&1&5&1\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&1&1&1&1&5
\end{matrix}$$

The compatibility matrices C_3

$$\begin{matrix}
7&1&1&0&0&1&1&1&1&0&0&0&0&1&1&1&0&0&0&0&0\\
1&7&1&1&0&0&1&1&1&1&0&0&0&0&1&1&1&0&0&0&0\\
1&1&7&1&1&0&0&1&1&1&1&0&0&0&0&1&1&1&0&0&0\\
0&1&1&7&1&0&0&0&1&1&1&1&0&0&0&0&1&1&0&0&0\\
0&0&1&1&7&0&0&0&0&1&1&1&0&0&0&0&0&1&0&0&0\\
1&0&0&0&0&7&1&1&0&0&0&0&1&1&1&0&0&0&0&0&0\\
1&1&0&0&0&1&7&1&1&0&0&0&1&1&1&1&0&0&1&0&0\\
1&1&1&0&0&1&1&7&1&1&0&0&0&1&1&1&1&0&1&1&0\\
1&1&1&1&0&0&1&1&7&1&1&0&0&0&1&1&1&1&1&1&1\\
0&1&1&1&1&0&0&1&1&7&1&1&0&0&0&1&1&1&0&1&1\\
0&0&1&1&1&0&0&0&1&1&7&1&0&0&0&0&1&1&0&0&1\\
0&0&0&1&1&0&0&0&0&1&1&7&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&7&1&1&0&0&0&0&0&0\\
1&0&0&0&0&1&1&1&0&0&0&0&1&7&1&1&0&0&1&0&0\\
1&1&0&0&0&1&1&1&1&0&0&0&1&1&7&1&1&0&1&1&0\\
1&1&1&0&0&0&1&1&1&1&0&0&0&1&1&7&1&1&1&1&1\\
0&1&1&1&0&0&0&1&1&1&1&0&0&0&1&1&7&1&1&1&1\\
0&0&1&1&1&0&0&0&1&1&1&1&0&0&0&1&1&7&0&1&1\\
0&0&0&0&0&0&1&1&1&0&0&0&0&1&1&1&1&0&7&1&1\\
0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&1&1&1&1&7&1\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&1&1&1&1&7
\end{matrix}$$

The compatibility matrices C_4

$$\begin{matrix}
7&2&1&0&0&1&2&2&1&0&0&0&0&1&1&1&0&0&0&0&0\\
2&7&2&1&0&0&1&2&2&1&0&0&0&0&1&1&1&0&0&0&0\\
1&2&7&2&1&0&0&1&2&2&1&0&0&0&0&1&1&1&0&0&0\\
0&1&2&7&2&0&0&0&1&2&2&1&0&0&0&0&1&1&0&0&0\\
0&0&1&2&7&0&0&0&0&1&2&2&0&0&0&0&0&1&0&0&0\\
1&0&0&0&0&7&2&1&0&0&0&0&2&2&1&0&0&0&0&0&0\\
2&1&0&0&0&2&7&2&1&0&0&0&1&2&2&1&0&0&1&0&0\\
2&2&1&0&0&1&2&7&2&1&0&0&0&1&2&2&1&0&1&1&0\\
1&2&2&1&0&0&1&2&7&2&1&0&0&0&1&2&2&1&1&1&1\\
0&1&2&2&1&0&0&1&2&7&2&1&0&0&0&1&2&2&0&1&1\\
0&0&1&2&2&0&0&0&1&2&7&2&0&0&0&0&1&2&0&0&1\\
0&0&0&1&2&0&0&0&0&1&2&7&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&2&1&0&0&0&0&0&7&2&1&0&0&0&0&0&0\\
1&0&0&0&0&2&2&1&0&0&0&0&2&7&2&1&0&0&1&0&0\\
1&1&0&0&0&1&2&2&1&0&0&0&1&2&7&2&1&0&2&1&0\\
1&1&1&0&0&0&1&2&2&1&0&0&0&1&2&7&2&1&2&2&1\\
0&1&1&1&0&0&0&1&2&2&1&0&0&0&1&2&7&2&1&2&2\\
0&0&1&1&1&0&0&0&1&2&2&1&0&0&0&1&2&7&0&1&2\\
0&0&0&0&0&0&1&1&1&0&0&0&0&1&2&2&1&0&7&2&1\\
0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&2&2&1&2&7&2\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&2&2&1&2&7
\end{matrix}$$

The compatibility matrices C_5

The Spontaneous Evolution Genetic Algorithm for Solving the Traveling Salesman Problem

William W. Hsu and Ching-Chi Hsu

Department of Computer Science and Information Engineering,
National Taiwan University
Taipei, Taiwan 106
Email: {r7526001, cchsu}@csie.ntu.edu.tw
Tel: 886-2-23917406

Abstract

In this paper, we present a new model of genetic algorithms, the Spontaneous Evolution Genetic Algorithm (SEGA) that incorporates both the concept of spontaneous generation and winner-take-all competition. It employs the idea of adding potential members during the search process, which keeps the evolution process boiling and thus preventing premature convergence. The winner-take-all competition preserves elitism during the search process. We used a learning scheme to preserve past results for future use in creating new candidates, which are the potential members, during the spontaneous generation phase. Experiments conducted on TSP with this new SEGA model proved that it produces better quality solution than bare GLS, LK local search, and simple SEGA (without learning memory).

1 INTRODUCTION

The importance of the traveling salesman problem (TSP) did not come from the need of the salesman in minimizing their travel routes, but in fact, from the need of many real world problems. Drilling of printed circuit boards, physical wiring of networks, vehicle routing, drilling of printed circuit boards, chronological sequence, and the double digest problem (DDP) from computational biology can be modeled into a TSP instance.

The traveling salesman problem (TSP) is closely related to the Hamiltonian-cycle problem, a salesman must visit N cities. Modeling the problem as a complete graph $G(V,E)$ with N vertices and $(N*(N-1))/2$ edges, which represents the cities and the routes respectively, we can say that the salesman wishes to make a tour, or Hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from. There is a cost $c(i,j)$ to travel from city i to city j, and the salesman wishes to minimize to total cost of traversing the tour, where total cost is the sum of the individual costs along the edges of the tour.

Mathematically, the traveling salesman problem can be formulated as finding a permutation π of the set {1,2,3,..,n} that minimizes the quantity:

$$C(p) = \sum_{i=1}^{n-1} d_{p(i),p(i+1)} + d_{p(n),p(1)} \quad (1)$$

where d_{ij} denotes the distance between city i and city j. Formula (1) is referred to as the cost or distance of a tour. For genetic algorithms, it is called fitness function. Two different cases of TSP are studied: symmetric TSP (STSP) and asymmetric TSP (ATSP). In STSP, $d_{ij} = d_{ji}$ holds for all i and j. For the ATSP case, this condition is not satisfied. In this paper, attempts are focused on the STSP case only.

We can see that the TSP problem is simple to state but hard to solve. For a case with N cities, there will be $(N-1)!/2$ possible tours. For the symmetric 532-cities instance, there are more than 10^{1000} possible tours. From Gary and Johnson's [2] work, the traveling salesman problem is stated to belong to NP-complete class. Thus, a fast and exact algorithm for the traveling salesman problem is unlikely to exist.

2 GENETIC ALGORITHMS

Evolutionary computing (Soft computing) has been proven useful in solving NP-Complete and NP-Hard problems. For the traveling salesman problem, genetic algorithms have been a great success.

A genetic algorithm traditionally contains three types of operators: selection, crossover and mutation. A simple genetic algorithm executes as follows [16]:

1. Start with a randomly generated population of n x-bit chromosomes. These are the candidate solutions to the problem.

2. Calculate the fitness *F(x)* of each chromosome *x* in the population.
3. Repeat the following steps until *n* offspring have been created:
    a. Select a pair of chromosome playing the role as parents. The probability of an individual been selected is usually a function of fitness. The fitter the individual is, the more likely it will be selected to reproduce.
    b. With a probability P_c (the crossover rate), crossover the pair at a randomly chosen point to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents.
    c. Mutate the two offspring at each locus with probability P_m (the mutation rate) and place the resulting chromosomes in the new population
4. Replace the current population with the new population.
5. Go to step 2.

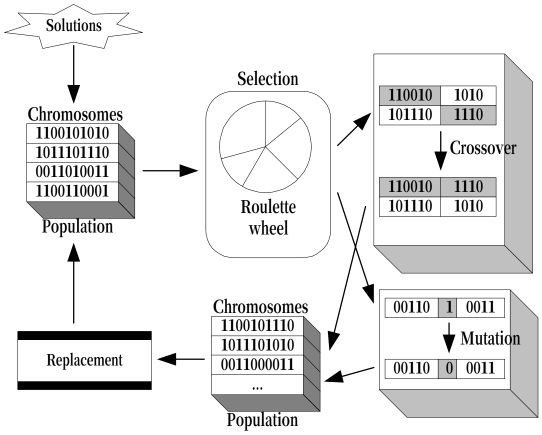


Figure 1: Traditional Genetic Algorithm

Each iteration of this process is called a generation. The entire set of generations is called a run. Usually, the desired solution will be found at the end of a run having the best fitness value.

3 THE SPONTANEOUS EVOLUTION GENETIC ALGORITHM

Although genetic algorithms has been proven to be successful in attacking the Traveling Salesman Problem, there is one major improvement that can be made: the problem of premature convergence in genetic algorithms. In this work, a new genetic algorithm model incorporating spontaneous generations spawned from the spawning pool, which is assisted by a learning memory to keep track of past events plus a winner-take-all competence within the individual members to provide top elitism is proposed. Since in this model, a new member generated from the spawning pool with reference to the learning memory is introduced into the population every generation, we will call this algorithm the Spontaneous Evolution Genetic Algorithm (SEGA). The flow of this algorithm is shown in Figure2. Detail descriptions of this algorithm will be given in the following sections.

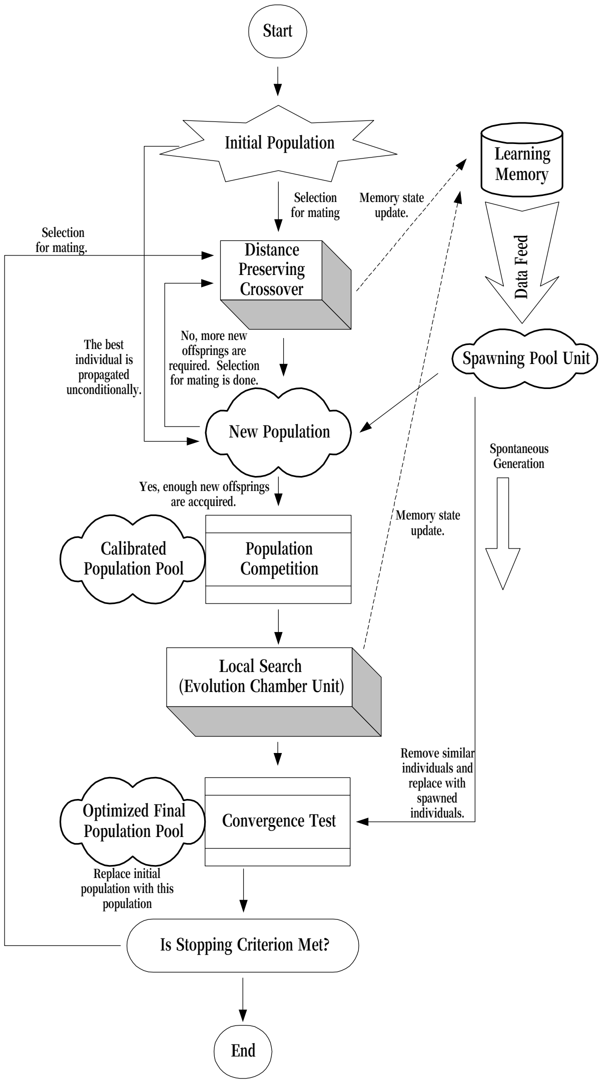

Start
Initial Population
Learning Memory
Selection for mating
Memory state update.
Selection for mating.
Distance Preserving Crossover
Data Feed
The best individual is propagated unconditionally.
No, more new offsprings are required. Selection for mating is done.
Spawning Pool Unit
New Population
Spontaneous Generation
Yes, enough new offsprings are acquired.
Memory state update.
Calibrated Population Pool
Population Competition
Local Search (Evolution Chamber Unit)
Remove similar individuals and replace with spawned individuals.
Optimized Final Population Pool
Convergence Test
Replace initial population with this population
Is Stopping Criterion Met?
End

Figure 2: The Spontaneous Evolution Genetic Algorithm

3.1 POPULATION INITIALIZATION POLICY

Yoichi and Nobuo [20] have proposed the use of geometric properties of points in construction of the initial population. They stated that the Visiting Order Restriction Theorem (VORT), proved by Flood [4], gives a necessary condition for the shortest tour of TSP on the

Euclidean plane by using the convex hull. VORT is stated as follows:

> Let *S* be the set of given cities in the plane, *C* be the convex hull of *S*, *V* be the set of cities on the boundary of *C*, *P* be the shortest tour of *S*. Let *F* be the polygons that are made up by tracing the cities of *V* according to the visiting order in the shortest path *P*. Then polygon *F* is equivalent to the convex hull *C.*

Based on this concept, Hsu [8] has developed a general initialialization technique using the 1-Tree heuristic (see section 3.5) plus the convex hull optimization, and proved that this technique produces better initial solution candidates. In SEGA, we have chosen to use 1T+CH as our initialization policy.

3.2 GENETIC EVOLUTION STRUCTURE

We have adopted the GLS [15] architecture and made modifications to it. In the case of STSP, the GLS approach starts with the nearest neighbor (NN) [10] initialization policy and uses the variable *k*-change heuristic suggested by Lin and Kernighan [11] for local search. Merz and Freisleben [15] pointed out that for most instances defined in the TSPLIB [18], the average percentage excess is about 2% above the optimum, although there are instances with topographical structures for which the results are much worse, such as the instance *fl3795* [9].

```
Algorithm SEGA
{
  Initialize population P with a construction heuristic;
  Repeat
  {
    For a = 1 to #crossovers
    {
      Select two parents $i_a$ and $i_b \in P$ randomly;
      $i_c$ := RDPX($i_a$,$i_b$);
      Add individual $i_c$ to P;
      If $i_c$ is better than $i_a$ and $i_b$ then
        Update_Learning_Memory($i_c$);
    }
    Spontaneous-Generate(i);
    Add I to P;
    Competition(P);
    Convergence_Detection(P);
  }
  Until P converged;
}
```

Figure 3: Pseudocode of SEGA

The mutation operator used in the GLS is called the non-sqeuential four change [11]. Since in the Lin-Kernighan heuristic performs only sequential changes, the effects of this kind of mutation cannot be reversed in a single step, and there is a high probability for ending up with a new solution after the local search phase. Due to that only four edges are exchanged, most information coded in the chromosome will be preserved.

Our modification of the GLS algorithm into SEGA is shown in Figure 3. Since we use *5-opt* sequential and non-sequential move for our local search heuristic, the non-sequential four changes will also be included (by using combinations of *2-opt* and *3-opt* changes that disconnects and then reconnects the disconnected tours). Thus, we did not incorporate the mutation operator into our algorithm because it is embedded in the local search procedure. From this aspect, our genetic algorithm is much simpler.

3.3 SELECTION OPERATOR

The selection operation used in our method is the uniform distributed selection operator. Every individual in the population has the same probability to be selected for mating. This method was considered because the population sized we used for our experiments were quite small, ranging from 10 to 20 individuals. If we bias the selection in anyway (by using rank selection or fitness proportionate selection [16]), the population diversity will decrease quickly and the population will converge prematurely.

After each mating, the individuals are restored back into the population. This implies that they may be reselected again for mating. By using this approach, we have a high probability to create a more diverse next generation.

3.4 CROSSOVER OPERATOR

The distance preserving crossover (DPX) [15] can preserve common information of the parents into the offspring, and can lead to promising regions of the search space without degrading to an uncontrolled random walk. The distance of two possible tours is a precise measure of the level similarity between them. This metric is defined as follows [13]: Let $G=(V, E)$ be the graph of a given TSP instance and T_1, T_2 be two feasible tours, then the distance D between the tours is defined as

$$D(T_1,T_2) = \left|\{e \in E \mid e \in T_1 \wedge e \notin T_2\}\right| \qquad (2)$$

The motivation of the DPX operator was from the analysis of TSP search space by Boese [1]. Boese's observation on the symmetric 532-cities instance of Padberg and Rinaldi [17] shows that the optimum lies more or less in a single valley of near-optimum local minima and the average distance between these near-optimum solutions is similar to their distance to the optimum. Another motivation is that by identifying the edges that are not shared by the parents, the search will be focused on particular regions of the search space.

DPX works as follows: the content of the first parent is copied to the offspring and all edges that are not in common with the other parent are deleted. The resulting fragments of the broken tour are reconnected without using any of the edges that are contained in only one of the parent. A greedy reconnection procedure is employed to achieve this: if the edge (*i,k*) has been destroyed, the nearest available neighbor *k* of *i* among the endpoints of the remaining tour fragments is taken and the edge (*i,k*) is added to the tour, provide that (*i,k*) is not contained in one of the two parents. This process continues until all of the tour fragments are reconnected to form a feasible tour. Thus, the child has equal distance to both parents and this distance is equal to the parents themselves. Figure 4 illustrates this process.

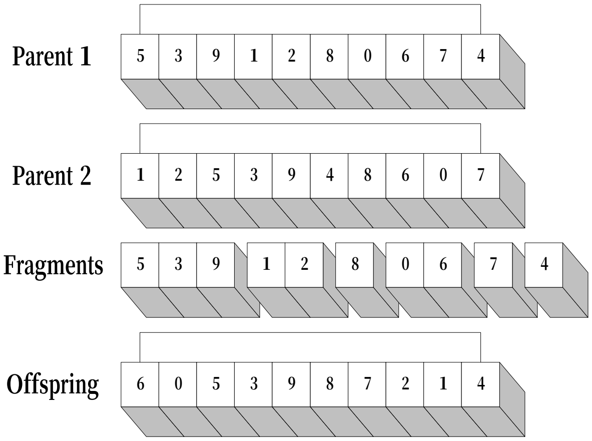


Figure 4: The DPX Crossover

Our implementation of the DPX crossover relaxes the constraint that the distance between the offspring and the parents and the distance of the parents themselves has to be the same. We call this implementation Relaxed DPX (RDPX). In addition, we did not use the greedy reconnection strategy for reconnecting the tour fragments. We used random reconnection strategy to form the feasible offspring tour. Our relaxation has caused the distance between the offspring and the parents are at least as small as the distance between the parents themselves. This approach was used for two reasons. First, greedy algorithms are usually time consuming and may not produce optimal configurations most of the time. Second, this implementation is simple and saves a great deal of computational time because it is purely stochastic.

3.5 1-TREE HEURISTIC

Selecting the *N* closest cities as a candidate set for a specific city to connect to may be inappropriate. Padberg and Rinaldi [17] has pointed out one of the links in the optimal tour of *att532* is the 22nd nearest neighbor city for one of its endpoint. However, selecting 22 candidates for each city may be inappropriate since it results in a substantial increase in search time.

The candidate set manipulation proposed by Helsgaun [5] used the minimum *1-tree* to select the candidate cities for each city. A *1-tree* is defined as follows:

> *A **1-tree** for a graph G=(N,E) is a spanning tree on the node set N-{1} combined with two edges from E incident to node 1.*

The choice of node 1 as a special node is arbitrary. It can be seen that a 1-tree is not really a tree since it contains a cycle as shown in Figure 5.

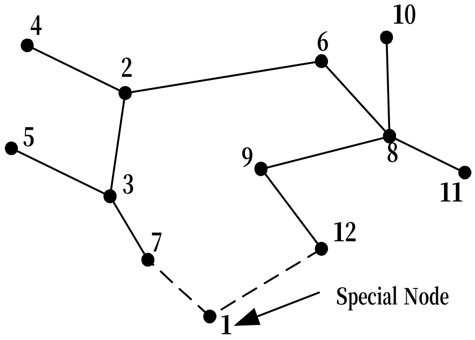


Figure 5: An example of a 1-tree

Usually, a minimum spanning tree contains many edges in common with an optimal tour. A minimum 1-tree is even more suitable in measuring the possibility of an edges being in the optimal tour. This is because the optimal tour usually contains 70 to 80 percent of the edges of a minimum 1-tree. The probability of an edge being in the optimal tour is measured by the nearness of the edge to the minimum 1-tree. Edges that belong or nearly belong to the minimum 1-tree has a high chance of belonging to the optimal tour. Conversely, edges that are far from belonging to the minimum 1-tree have a low probability of belong to the optimal tour. Formally, this measure of nearness is defined as follows:

> *Let T be a minimum 1-tree of length L(T) and let T+(i,j) denote a minimum 1-tree required to contain the edge (i,j). Then the a-nearness of an edge (i,j) is defined as the quantity:*

$$a(i,j) = L(T^{+}(i,j)) - L(T) \tag{2}$$

That is, given the length of any minimum 1-tree, the *a*-nearness of an edge is the increase of length when a minimum 1-tree is required to contain this edge. This is represented by Formula (2).

The *a*-nearness is a systematic way to identify edges that are potent to be included in the optimal tour. These promising edges can help form the candidate set, e.g., consist of the *k a*-nearest edges incident to each node. The use of *a*-nearness measures for specifying the candidate set is much better than using nearest neighbors. Usually, the candidate set will be smaller without degrading the final solution. The efficiency of this method has a time complexity of $O(n^2)$ and a space complexity of O(n) [2, 19].

3.6 LOCAL SEARCH OPTIMIZATION

In our algorithm, we used sequential *5-opt* moves for our basic moves. This approach attempts to ensure *2*, *3*, *4* and *5-opt*. The moves considered are now sequences of one or more *5-opt* move, and the construction of a move halts when a feasible tour improvement is detected. Selecting *5-opt* for our basic move was due to observations made by Christofides and Eilon [3]. They pointed out that *5-opt* should be expected to yield a relatively superior improvement over *4-opt* compared with the improvement of applying *4-opt* over *3-opt*.

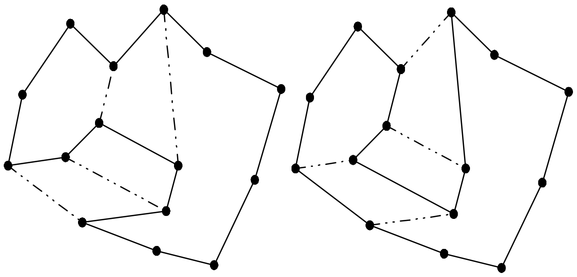


Figure 6: A double bridge.

Shown in Figure 6 is the double bridge phenomenon, where a peninsula hops around in the tour. Only non-sequential moves can allow these kind changes. These kind of changes are also called kicking moves, which was proposed by Martin, Otto and Felten [14]. We used two methods to obtain the same results of the kicking move:

1. We did a *2-opt* move that disconnects the tour by producing two cycles and then a *2-opt* or *3-opt* move that produces a feasible tour by reconnecting the cycles together.
2. We did a *3-opt* move that disconnects the tour by producing two cycles and then a *2-opt* move that produces a feasible tour by joining these two cycles together.

Using this approach, we can kick the tour in less time than using non-sequential *4-opt* moves, and the chances of finding good kicks are greater [5].

4 LEARNING FOR SPONTANEOUS EVOLUTION

This memory cells idea came from Lo and Hsu's [12] Annealing Learning Scheme (ALS). They incorporated a memory unit that will alter the probability of acceptance of their annealing process. This allows past search results to contribute their information for current use. If we can somehow allow information from the search process guide our spontaneous generation, we can create new members that have higher chance of evolving into optimal solutions. This prevents adding members to the population blindly by using initialization techniques that creates a member just like the beginning of the algorithm, which starts the search all over again.

4.1 MEMORY CELL: SMART EVOLUTION

The dimension of the memory cell is *N*N*, where *N* is the number of city. At the beginning of the algorithm, where the evolution process starts, we set all of the elements in the memory cells to zero. We use *M*(*i,j*) to represent the value of a specific memory cell position, and (*i,j*) itself to represent the link or edge from city *i* to city *j*.

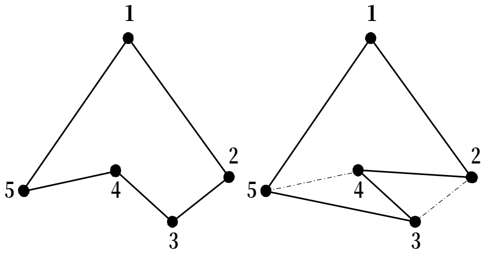


Figure 7: A 5-city Example

	1	2	3	4	5
1	0	0	0	0	1
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	0	1	0

	1	2	3	4	5
1	0	0	0	0	1
2	1	0	0	0	0
3	0	1	0	0	0
4	0	1	1	0	0
5	0	0	1	1	0

Figure 8: Learning Memory Configuration of Figure 7

Next, we must have a mechanism of updating datas into the memory cells. It is not difficult to verify that *M*(*i,j*) = 0 for all *i*=*j*, since that there will be no loop-backs. We use the following two rules to make changes to the memory cells:

1. If the individual *T* was improved by using the RDPX operator, then we consider that this chromosome is good and update the whole member's link to the memory.

   ∀edges (*i,j*) in *T*: *M*(*i,j*) = *M*(*i,j*) + 1

2. If a specific or a chain of *flip* operations improves the tour, then we update these edges in the change into the memory.

   ∀edges (*i,j*) involving in the *k-change* of *T*: *M*(*i,j*) = *M*(*i,j*) + 1

Consider the example shown in Figure 7. Assume some RDPX operation was conducted on the two parents producing the offspring shown in Figure 7 (left), the memory will be updated according to rule 1 and the results shown in Figure 8 (left). Following by is a local search *k-opt* move, which changes 2 edges (2,3) and (4,5) to (2,4) and (3,5) respectively (shown in Figure 7 (right)), the memory will be updated using rule 2 and the final result is shown in Figure 8 (right).

Using this simple algorithm, we can effectively keep track of what edges contribute in leading to the optimal solution. We neither waste the previous search efforts nor keep excessive data. In our case, our destined problem is to solve the symmetric TSP where edge (*i,j*) has the same meaning as edge (*j,i*). When we update *M*(*i,j*), we update *M*(*j,i*) correspondently too. The result is that the learning memory is reflexive, i.e., *M*(*i,j*) = *M*(*j,i*)**.** This algorithm has a time complexity of O(n) and a space complexity of O(n^2).

4.2 SPONTANEOUS GENERATIONS

With our learning memory constructed during the search process, we need a method to let the spawning pool know how to generate spontaneous members. Our approach uses greedy algorithm with some randomization.

	1	2	3	4	5
1	0	6	1	0	3
2	6	0	1	2	5
3	1	1	0	4	2
4	0	2	4	0	0
5	3	5	2	0	0

Figure 9: A Learning Memory Configuration

Consider the learning memory configuration shown in Figure 9. Assume that we now want to generate a spontaneous member from this configuration. The following steps for spontaneous member construction are listed as follows:

$$j \notin \textbf{selected}, \quad p((i,j)) = \frac{M(i,j)}{\sum_{\forall j \notin selected} M(i,j)}$$

1. Select a random city *i*, as the starting node. Mark city *i* as selected.
2. Obtain the column sum *S* of city *i* from the learning memory. If *S*=0, then select the next city *j* randomly and goto step 4, otherwise select city *j* with step 3.
3. The probability of city *j*, i.e. edge (*i,j*), being selected as next city is:
4. Mark city *j* as selected and let *i*=*j*. If there are still cities unselected, then goto step 2.

Let us begin with city 3. The column sum of city 3 is 1+1+4+2=8. The chance of selecting city 1, 2, 4 and 5 are1/8, 1/8, 4/8, and 2/8 respectively. Now assume that we select city 4 as the next city. The selected list is now (3,4) and the tour list is (3,4). For city 4, the column sum is 0+2+0=2 (since city 3 and 4 are marked selected). The probability of selecting city 1,2, and 5 are 0/2, 2/2, and 0/2 respectively. It is intuitively that we will select city 2 as the next city. The tour list is now (3,4,2) and the marked city list is (3,4,2). We continue this process until all the cities are selected into the tour list forming a feasible tour.

5 INDIVIDUAL COMPETITION

If our initial population size is *N*, then we will have a population size of 2*N* + 1 now (*N* individuals created from crossover and one individual created spontaneously from the spawning pool). The major objective here is to simulate the cruel environment: the fittest one gets all the resource for survival, i.e., it gets to perform local search. Hsu [7] has proposed this competition algorithm (shown in Figure 10), and proved that GA using competition yields better solution.

```
Algorithm Competition
{
   Sort population P in non-decreasing order;

   Do
   {
      If an individual has fitness in common with some other
      individuals then
         Replace this individual with a spontaneous
         generated individual;
   } While no two individuals of P have the same fitness

   Sort this new calibrated population P in non-decreasing
   order;

   Starting from the most fitted individual i and do
   {
      $i_n$ = Local-Search(i);
      If $i_n$ is better than i then:
         Replace i with $i_n$ and break;
      Else
         Select the next fitted individual for local search;
   }
   Select the individuals to survive for the next generation;
}
```

Figure 10: The Competition Algorithm

The local search of the SEGA occurs only in this step. Starting from the fittest individual, we perform local search on this individual. If the result of the local search improves this individual, we replace this individual by the one produced by local search procedure. Otherwise, we choose the next individual and apply local search to it. We continue this process until an improvement is detected or all of the individuals are checked. We also call this algorithm the winner-take-all competition since that only the best of the best member gets to perform local search.

Finally, we select the *N* most fitted individual for survival and let them become the new initial population for the next generation of evolution. Using the individual competition approach guarantees in finding a better solution or preserve the current best solution found so far, i.e., elitism.

6 EXPERIMENT RESULTS

The experiment conducted in this section uses a population size of 20. 1-Tree and convex hull optimization (1T+CH) scheme was used for population initialization. 20 trials were executed for each test case and each trial is executed for 100 generations. The recorded data was the average of these 20 trials. The machine used is Intel Pentium III 500MHz with 256M of memory running Mandrake Linux v7.2.

The following schemes will be compared:

- **BARE**: No spawning pool, no spontaneous generations and no learning memory are used. It is just identical to bare genetic local search (GLS) algorithm.
- **LS**: Local search method by Hsu [7].
- **Simple SEGA**: The spawning pool was used to generate spontaneous members, but without any learning memory. This is actually blind initiation (random) of a member introduced into every generation of GLS.
- **SEGA**: The model proposed in this paper.

Table 1: Average Execution Time

Problem Name	Method (Best Gap)			
	BARE	LS	Simple	SEGA
att48	0.000%	0.000%	0.000%	0.000%
eil51	0.000%	N/A	0.000%	0.000%
kroA100	0.000%	N/A	0.000%	0.000%
d198	N/A	0.000%	N/A	N/A
lin318	0.123%	0.00%	0.000%	0.000%
pcb442	0.005%	N/A	0.000%	0.000%
att532	0.028%	0.000%	0.000%	0.000%
rat783	0.004%	0.000%	0.002%	0.000%
vm1084	0.041%	N/A	0.018%	0.011%
d1291	0.055%	0.000%	0.040%	0.000%
fl1577	0.007%	0.010%	0.012%	0.002%

From Table 1, we can see that LS is the fastest method. This can be explained because BARE, simple SEGA, and SEGA are actually conducting multiple LS during each generation. BARE is fast but yields poor quality solutions because of the premature convergence property of genetic algorithms. From the three genetic algorithm mutants, we can see that SEGA outperforms BARE and simple SEGA (although it is slower that BARE, it yield better solutions). This is because it is able to find better solution and then converge (thought it may not be an optimal solution).

Table 2: Comparasion of Average Solution Gap

Problem Name	Method (AVG Gap)			
	BARE	LS	Simple	SEGA
att48	0.000%	0.000%	0.000%	0.000%
eil51	0.000%	N/A	0.000%	0.000%
kroA100	0.000%	N/A	0.000%	0.000%
d198	N/A	0.050%	N/A	N/A
lin318	0.192%	0.310%	0.000%	0.000%
pcb442	0.210%	N/A	0.002%	0.000%
att532	0.185%	0.200%	0.009%	0.000%
rat783	0.362%	0.040%	0.186%	0.002%
vm1084	1.467%	N/A	0.111%	0.012%
d1291	0.027%	0.340%	0.023%	0.017%
fl1577	0.058%	0.920%	0.036%	0.012%

Table 3: Best Solution' s Gap Obtained

Problem Name	Method (Avg. Exec Time in Seconds)			
	BARE	LS[1]	Simple	SEGA
att48	0.152	0.180	0.162	0.096
eil51	0.147	N/A[2]	0.175	0.106
kroA100	0.261	N/A	0.222	0.263
d198	N/A	1.430	N/A	N/A
lin318	3.260	2.120	2.912	3.105
pcb442	10.007	N/A	19.864	5.226
att532	8.925	5.120	15.554	9.513
rat783	10.503	6.870	23.102	12.295
vm1084	25.174	N/A	419.495	387.872
d1291	44.62	19.100	1452.523	767.235
fl1577	1089.78	57.670	3349.235	1513.630

Both tables 2 and 3 shows that SEGA provides the best solution quality, both on average and best case. Comparing to SEGA, the large execution time of simple SEGA is due to that a new search is started when a spontaneous member is introduced. This is because this member is initialized randomly and can be anywhere in the search space Besides the search process has a very high probability to end where it has started before adding this spontaneous member. This is why we did not use

---

[1] Result obtained from Hsu [7].

[2] Fields containing N/A represent that experiments for those instances are not conducted.

random (blind) generation of the spontaneous member. Although without using spawning pool and learning memory has a faster execution time, the drawback is that the chance for it to find optimal solution is lessen.

7 CONCLUSION

We have proposed a successful new genetic algorithm model, the Spontaneous Evolution Genetic Algorithm (SEGA). This model has three major capabilities. First, it is able to prevent premature convergence by introducing new members into the population. Second, this new SEGA does not generate these new members blindly. It does so by referencing to the learning memory. This learning memory assists the spawning pool, where the new member is produced, in building feasible and good candidate tours with in formations from past search trials. This process is called a spontaneous generation. Third, the local search operator was carefully used in combinations with mutation operator, i.e., mutation is performed during local search. With this assumption, we removed the mutation step in GLS. In addition, by allowing only a single member to enter the evolution chamber for local search sped up the algorithm.

Proven that the spawning pool unit (spontaneous generation) and evolution chamber unit (winner-take-all competition) are successful, this model can be applied to other problems, such as graph partitioning, knapsack problem, and vehicle routing problems.

Reference

[1] K. Boese. “Cost verses Distance in the Traveling Salesman Problem”. *Technical Report Tr-950018*, UCLA CS Department, 1995.

[2] G. Carpaneto, M. Fichetti, and P. Toth. “New Lower Bounds for the Symmetric Traveling Salesman Problem”. *Math. Programming*, 45, pp. 233-254, 1989.

[3] N. Christofides and S. Eilon. “Algorithms for Large-scale Traveling Salesman Problem”. *Operations Research*, vol. 23, pp. 511-518, 1972.

[4] M. M. Flood. “The traveling salesman problem”. *Operations Research*,. vol. 4, pp. 61-75, 1956.

[5] K. Helsgaun. “An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic”. *DATALOGISKE SKRIFTER (Writings on Computer Science)*, no. 81, 1998.

[6] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI:Univ. Michigan Press, 1975.

[7] W. W. Hsu and C. C. Hsu (2000). Applying Competition to Genetic Local Search for Solving the Traveling Salesman Problem, *Proceedings of the Fifth Conference on Artificial Intelligence and Applications (TAAI 2000)*, pp. 436-443. Taipei, Taiwan.

[8] W. W. Hsu and C. C. Hsu (2001), Hybrid Initialization Techniques on Local Search Heuristic for Solving the Traveling Salesman Problem, *Submitted to Congress on Evolutionary Computation 2001*.

[9] D. S. Johnson and L. A. McGeoch. “The Traveling Salesman Problem: A Case Study in Local Optimization”. *Local Search in Combinatorial Optimization*, Wiley and Sons, New York, 1996.

[10] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. New York: Wiley and Sons, 1985.

[11] S. Lin, B. W. Kernighan. “An Effective Heuristic Algorithm for the Traveling-Salesman Problem”. *Operations Research 21*, vol. 21, 498-516, 1973.

[12] C. C. Lo and C. C. Hsu. “Annealing Framework with Learning Memory”. *IEEE Transactions on System, Man, and Cybernetics*, part A, vol. 28, no 5, pp1-13, IEEE Press 1998.

[13] K.-T. Mak and A. J. Morton. “Distance between Traveling Salesman Tours”. *Discrete Applied Mathematics and Combinatorual Operations Research and Computer Science*, vol. 58, pp. 281-291, 1995.

[14] O. Martin, S. W. Otto, and E. W. Felten. “Large-step Markov Chains for the TSP Incorporating Local Search Heuristics”. *Operations Research Letters*, 11, pp. 219-224, 1992.

[15] P. Merz and B. Freisleben. "Genetic Local Search for the TSP: New Results". Proceedings of the 1997 *IEEE International Conference on Evolutionary Computation*, IEEE Press, pp. 159-164, 1997.

[16] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[17] M. Padberg and G. Rinaldi. “Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut”. *Operations Research Letters*, vol. 6, pp.1-7, 1987.

[18] G. Reinelt. “TSPLIB - A Traveling Salesman Problem Library”. *ORSA Journal on Computing*, vol. 3, no. 4, pp. 365-384, 1991.

[19] T. Volgenant and R. Jonker. “The Symmetric Traveling Salesman Problem and Edge Exchanges I Miminal 1-trees”. *Eur. J. Oper. Res.*, 12, pp. 394-403, 1983.

[20] T. Yoichi and F. Nobuo. “An Improved Genetic Algorithm Using the convex Hull for Traveling Salesman Problem”. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, pp. 2279-2284, 1998.

Independent Sampling Genetic Algorithms

Chien-Feng Huang
Center for the Study of Complex Systems, 4477 Randall Lab.
University of Michigan
Ann Arbor, MI 48109
cfhuang@engin.umich.edu
(734)763-3323

Abstract

Premature convergence is the loss of diversity in the population that has long been recognized as one crucial factor that hinders the efficacy of crossover. In this paper, a strategy for independent sampling of building blocks is proposed in order to nicely implement implicit parallelism. Based on this methodology, we developed a modified version of GA: independent sampling genetic algorithms (ISGAs). Simply stated, each individual independently samples candidate schemata and creates population diversity in the first phase; subsequently we allow individuals to actively select their mates for reproduction. We will present experimental results on two benchmark problems, "Royal Road" functions of 64-bits and bounded deception of 30-bits, to show how the performance of GAs can be improved through the proposed approach.

1 INTRODUCTION

Genetic algorithms (GAs) have been successfully applied to several difficult search and optimization problems in science and engineering. One major source of the power of GAs is derived from so-called implicit parallelism (Holland, 1975), i.e., the simultaneous allocation of search effort to many regions of the search space. A perfect implementation of implicit parallelism implies that a large number of different short, low-order schemata of high fitness are sampled in parallel, thus conferring enough diversity of fundamental building blocks for crossover operators to combine them to form more highly-fit, complicated building blocks. However, traditional GAs suffer from premature convergence (Goldberg, 1989) where considerable fixation occurs at certain schemata of suboptimal regions before attaining more advancement. Among examples of premature convergence, hitchhiking (Das, & Whitley, 1991; Mitchell, 1996) has been identified as a major hindrance, which limits implicit parallelism by reducing the sampling frequency of various beneficial building blocks. In short, non-relevant alleles hitchhiking on certain schemata could propagate to the next generation and drown out other potentially favorable building blocks, thus preventing independent sampling of building blocks. Consequently, the efficacy of crossover in combining building blocks is restricted by the resulting loss of desired population diversity.

Mitchell, Holland and Forrest (1994) considered a so-called "idealized genetic algorithm" (IGA) that allows each individual to evolve completely independently; thus new samples are given independently to each schema region and hitchhiking is suppressed. Then under the assumption that the IGA has the knowledge of the desired schemata in advance, they derived a lower bound for the number of function evaluations that the IGA will need to find the optimum of Royal Road function R_1 (Mitchell, Forrest, & Holland, 1992).

However, the IGA is impractical because it requires the exact knowledge of desired schemata ahead of time. Partially motivated by the idea of the IGA, we propose a more robust GA that proceeds in two phases: the "independent sampling phase" and the "breeding phase". In the independent sampling phase, we design a core scheme, named the "Building Block Detecting Strategy" (BBDS), to extract relevant building block information of a fitness landscape. In this way, an individual is able to sequentially construct more highly-fit partial solutions. For Royal Road R_1, the global optimum can be attained easily. For other more complicated fitness landscapes, we allow a number of individuals to adopt the BBDS and independently evolve in parallel so that each schema region can be given samples independently. During this phase, the population is expected to be seeded with promising genetic material. Then follows the breeding phase, in which individuals are paired for breeding based on two mate selection schemes (Huang, 2001): individuals being assigned mates by natural selection only and individuals being allowed to actively choose their mates. In the latter case, individuals are able to distinguish candidate mates that have the same fitness yet have different string structures, which may lead to quite different performance after crossover. This is not achievable by natural selection alone since it assigns individuals of the same fitness the same probability for being mates, without explicitly taking into account string structures. In short, in the breeding phase individuals manage to construct even more promising schemata through the recombination of highly-fit building blocks found in the first phase. Due to the characteristic of independent sampling of building blocks that distinguishes the proposed GAs from conventional GAs, we name this type of GA *independent sampling genetic algorithms* (ISGAs).

2 LITERATURE REVIEW

In GA research, extensive attention has been paid to how to alleviate premature convergence. An example is the class of parallel GAs (PGAs) that are developed to degrade centralized selection control used in simple GAs in order to accommodate more population diversity. Among these PGAs, the "fine-grained" type (Cantú-Paz, 1997) is an idealized model that allows only one individual to evolve in each deme and thereby implements the decentralization of selection scheme to the maximum degree. Mühlenbein (1991) used a local hillclimbing algorithm to refine the individuals in his fine-grained PGAs along with a mating strategy based on population structure and the empirical results showed that his PGA is an effective optimization tool.

The independent sampling phase of ISGAs is similar to the fine-grained PGAs in (Mühlenbein, 1991) in the sense that each individual evolves autonomously, although ISGAs do not adopt the population structure. The second distinction is that Mühlenbein's fine-grained PGAs process strings in a homogeneous fashion. An initial population is randomly generated. Then in every cycle each individual does local hillclimbing, and creates the next population by mating with a partner in its neighborhood and replacing parents if offspring are better. By contrast, ISGAs partition the genetic processing into two phases: the independent sampling phase and the breeding phase as described in the preceding section. Third, the approach employed by each individual for improvement in ISGAs is different from that of the PGAs. During the independent sampling phase of ISGAs, in each cycle, through the BBDS, each individual attempts to extract relevant information of potential building blocks whenever its fitness increases. Then, based on the schema information accumulated, individuals continue to construct more complicated building blocks. However, the individuals of Mühlenbein's PGAs adopt a local hillclimbing algorithm that does not manage to extract relevant information of potential schemata.

The motivation of the two-phased ISGAs was partially from the "messy genetic algorithms (mGAs)" in (Goldberg, Korb, & Deb, 1989; Goldberg, Deb, Kargupta, & Harik, 1993). The two stages employed in the mGAs are "primordial phase" and "juxtapositional phase", in which the mGAs first emphasize candidate building blocks based on the guess at the order k of small schemata, then juxtaposing them to build up global optima in the second phase by "cut" and "splice" operators. However, in the first phase, the mGAs still adopt centralized selection to emphasize some candidate schemata; this in turn results in the loss of samples of other potentially promising schemata. By contrast, ISGAs manage to postpone the emphasis of candidate building blocks to the latter stage, and highlight the feature of independent sampling of building blocks to suppress hitchhiking in the first phase. As a result, population is more diverse and implicit parallelism can be fulfilled to a larger degree. Thereafter, during the second phase, ISGAs implement population breeding through two mate selection schemes as discussed in the preceding section. In this way, we may examine if the results obtained for the ISGAs are consistent with what has been done for simple serial GAs in (Huang, 2001).

In the following sections, we present the key components of ISGAs in detail and show the comparisons between the experimental results of the ISGAs and those of several other GAs on two benchmark test functions.

3 COMPONENTS OF ISGAS

ISGAs are divided into two phases: the independent sampling phase and the breeding phase. We describe them as follows.

3.1 INDEPENDENT SAMPLING PHASE

To implement independent sampling of various building blocks, a number of strings are allowed to evolve in parallel and each individual searches for a possible evolutionary path entirely independent of others.

In this paper, we develop a new searching strategy, Building Block Detecting Strategy (BBDS), for each individual to evolve based on the accumulated knowledge for potentially useful building blocks. The idea is to allow each individual to probe valuable information concerning beneficial schemata through testing its fitness increase since each time a fitness increase of a string could come from the presence of useful building blocks on it. In short, by systematically testing each bit to examine whether this bit is associated with the fitness increase during each cycle, a cluster of bits constituting potentially beneficial schemata will be uncovered. Iterating this process guarantees the formation of longer and longer candidate building blocks.

The operation of BBDS on a string can be described as follows.

1. Generate an empty set for collecting genes of candidate schemata and create an initial string with uniform probability for each bit until its fitness exceeds 0. (Record the current fitness as *Fit*.)

2. Except the genes of candidate schemata collected, from left to right, successively flip all the other bits, one at a time, and evaluate the resulting string. If the resulting fitness is less than *Fit*, record this bit's position and original value as a gene of candidate schemata.

3. Except the genes recorded, randomly generate all the other bits of the string until the resulting string's fitness exceeds *Fit*. Replace *Fit* by the new fitness.

4. Go to steps 2 and 3 until some end criterion.

The idea of this strategy is that the cooperation of certain genes (bits) makes for good fitness. Once these genes come in sight simultaneously, they contribute a fitness increase to the string containing them; thus any loss of one of these genes leads to the fitness decrease of the string. This is essentially what step 2 does and after this step we should be able to collect a set of genes of candidate schemata. Then at step 3, we keep the collected genes of candidate schemata fixed and randomly generate other bits, awaiting other building blocks to appear and bring forth another fitness increase.

However, the step 2 in this strategy only emphasizes the fitness drop due to a bit-flip. It ignores the possibility that the same bit-flip leads to a new fitness rise because many loci could interact in an extremely nonlinear fashion. To take this into account, the second version of BBDS is introduced through the change of step 2 as follows.

Step 2. Except the genes of candidate schemata collected, from left to right, successively flip all the other bits, one at a time, and evaluate the resulting string. If the resulting fitness is less than *Fit*, record this bit's position and original value as a gene of candidate schemata. If the resulting fitness exceeds *Fit*, substitute this bit's "new" value for the old value, replace *Fit* by this new fitness, record this bit's position and "new" value as a gene of candidate schemata, and re-execute this step.

Because this version of BBDS takes into consideration the fitness increase resulted from bit-flips, it is expected to take less time for detecting. Several empirical results so far support this reasoning (for example, the experimental results of these two versions on Royal Road functions shown in the next section).

Other versions of BBDS are of course possible. For example, in step 2, if a bit-flip results in a fitness increase, it can be recorded as a gene of candidate schemata, and the procedure continues to test the residual bits yet without completely travelling back to the first bit to re-examine each bit. However, the empirical results obtained thus far indicate that the performance of this alternative is quite similar to that of the second version. More experimental results are needed to distinguish the difference between them. In this paper, we present the results obtained based on the first and second versions of BBDS.

The overall implementation of the independent sampling phase of ISGAs is through the proposed BBDS to get autonomous evolution of each string until all individuals in the population have reached some end criterion.

In section 4, we will present an analysis of the BBDSs on two types of idealized test functions: "Royal Road" functions (non-deceptive) and problems of bounded deception (deceptive).

3.2 BREEDING PHASE

After the independent sampling phase, individuals independently build up their own evolutionary avenues by various building blocks. Hence the population is expected to contain diverse beneficial schemata and premature convergence is alleviated to some degree. However, factors such as deception and incompatible schemata (i.e., two schemata have different bit values at common defining positions) still could lead individuals to arrive at sub-optimal regions of a fitness landscape. Since building blocks for some strings to leave sub-optimal regions may be embedded in other strings, the search for proper mating partners and then exploiting the building blocks on them are critical for overwhelming the difficulty of strings being trapped in undesired regions. In (Huang, 2001) the importance of mate selection has been investigated and the results showed that the GAs are able to improve their performance when the individuals are allowed to select mates to a larger degree.

In this paper, we adopt two mate selection schemes analyzed in (Huang, 2001) to breed the population: individuals being assigned mates by natural selection only and individuals being allowed to actively choose their mates. Since natural selection assigns strings of the same fitness the same probability for being parents, individuals of identical fitness yet distinct string structures are treated equally. This may result in significant loss of performance improvement after crossover. This issue is the major concern in (Huang, 2001) and we continue this research line to ISGAs in this paper.

We adopt the tournament selection scheme (Mitchell, 1996) as the role of natural selection and the mechanism for choosing mates in the breeding phase is as follows:

During each mating event, a binary tournament selection—with probability 1.0 the fitter of the two randomly sampled individuals is chosen—is run to pick out the first individual, then choosing the mate according to the following two different schemes:

A. Run the binary tournament selection again to choose the partner.

B. Run another two times of the binary tournament selection to choose two highly-fit candidate partners; then the one more dissimilar to the first individual is selected for mating.

The implementation of the breeding phase is through iterating each breeding cycle which consists of 1) Two parents are obtained based on the mate selection schemes above. 2) Two-point crossover operator (crossover rate 1.0) is applied to these parents. 3) Both parents are replaced with both offspring if any of the two offspring is better than them. Then steps 1, 2, and 3 are repeated until the population size is reached and this is a breeding cycle. (To give crossover its stiffest test, we turn off mutation for all the performance tests in this paper.)

In (Huang, 2001), the results showed that the mate selection scheme B outperforms scheme A in general, given the objective of finding the global optimum with minimum time. Since those results were obtained in simple GAs, we are concerned with whether this conclusion can be extended to the ISGAs as well.

Having described the components of ISGAs, we are now on the road to test their performance.

4 EXPERIMENTAL RESULTS

Two types of test functions are used for examining the performance of the ISGAs: "Royal Road" functions (non-deceptive) and problems of bounded deception (deceptive). The performance of some other approaches will be compared with that of the ISGAs as well.

4.1 PERFORMANCE ON ROYAL ROAD FUNCTIONS

The Royal Road functions designed by Mitchell, Forrest, and Holland (1992) were to investigate in more detail the validity of the *Building Block Hypothesis* (Holland, 1975; Goldberg, 1989), which implies that the performance of GAs largely depends on the efficacy of crossover to combine small, highly-fit schemata to form more complex, highly-fit schemata. The fitness landscape of Royal Road functions consists of two characteristics: the presence of short, low-order, highly-fit schemata and hierarchical structure which allows these small schemata to repeatedly construct more and more highly-fit schemata and eventually reach the global optimum. One example of this class is Royal Road R_1 whose fitness landscape is composed of eight consecutive building blocks of eight ones each. It is apparent that Royal Road R_1 is a non-deceptive function and it was expected that GAs perform quite well on such a fitness landscape due to the Building Block Hypothesis. However, Mitchell's experimental results indicated that the unsatisfactory GA performance on this function is primarily from hitchhiking phenomenon, one of possible causes of premature convergence.

In (Mitchell, 1995), the performance of the GA was further compared with those of three iterated hill-climbing searching algorithms: steepest-ascent hill-climbing (SAHC), next-ascent hill-climbing (NAHC) (Mühlenbein, 1991), and random-mutation hill-climbing (RMHC) (Forrest, & Mitchell, 1993). They performed 500 runs for the GA with population size 128 and 200 runs for each of the three hill-climbing algorithms, and reported that SAHC and NAHC never found the optimum within 256,000 function evaluations but the GA can attain the optimum in an average of 61,334 function evaluations. Moreover, RMHC found the optimum only in an average of 6179 function evaluations, nearly ten times faster than the GA.

We performed 1000 runs of the ISGA on Royal Road

R_1 and the end criterion is the moment for the global optimum being found. It turned out that for such a hierarchical, non-deceptive structure only an individual is needed, based on the building block detecting strategy discussed earlier, to serve for good performance. It actually found the optimum only in an average of 975 function evaluations for the first version of BBDS and 901 for the second version—more than six times faster than RMHC and sixty times faster than the GA.

These experimental results can be summarized in Table 1 in which the standard deviation is shown for each case as well.

Table 1: Experimental Results on R_1

	Function Evaluations to Optimum	
	Mean	Standard Deviation
GA	61334	32583
SAHC	>256,000	–
NAHC	>256,000	–
RMHC	6179	2630
BBDS v. 1	975	314
BBDS v. 2	901	309

To see why the structures aggregated by the BBDS are indeed potentially promising schemata, let us turn to the motivation of the BBDS, i.e., the IGA. On the idealized model Royal Road R_1, Mitchell et al. (1994) discussed the expected time for the IGA to construct all the eight building blocks for reaching the optimum and obtained the theoretical result of 696 function evaluations. In the process of BBDS version 1, when the first building block emerges in the string, 64 evaluations are required to detect it since we need to flip all the 64 bits, one at a time, and evaluate the resulting string. Similarly, as the second building block comes in sight, another 56 evaluations is required to detect it. By this reasoning, the total evaluations required for detecting the building blocks are 64+56+48+40+32+24+16=280. (It is not necessary to detect the final single block because the appearance of the final building block is at the same moment of the optimum being attained.) If two or more building blocks appear simultaneously, the evaluations for detecting will be less than 280, but this occurs with a rather small probability. Therefore, the sum of 696 function evaluations (the theoretic result obtained by Mitchell et al.) for constructing all the eight building blocks and 280 function evaluations for detecting these building blocks is 976. This is almost perfectly consistent with the result obtained for BBDS version 1 shown in Table 1. Thus, we can conclude that BBDS implements nearly the idealized GA on Royal Road R_1 in the sense that extra function evaluations are required to detect the building blocks. As for the performance of the second version of BBDS, since it adopts a more greedy method to detect the building blocks, it takes less time to attain the optimum than the first version does.

Another idealized model to test the power of BBDS is Royal Road R_2 (Forrest, & Mitchell, 1993). This function was designed to verify if the presence of intermediate “stepping stones” (intermediate-order higher-fitness schemata that result from combinations of the lower-order schemata, and that in turn can combine to form even higher-fitness schemata) can speed up GAs’ searching process. Forrest et al. (1993) found that if some intermediate stepping stones are much fitter than the primitive components, then hitchhiking problem becomes more severe and thus premature convergence slows down the discovery of some necessary schemata.

We summarize the results from two versions of the BBDS and those reported in (Forrest, & Mitchell, 1993) in Table 2.

Table 2: Experimental Results on R_2

	Function Evaluations to Optimum	
	Mean	Standard Deviation
GA	73563	40115
SAHC	>256,000	–
NAHC	>256,000	–
RMHC	6551	2998
BBDS v. 1	975	314
BBDS v. 2	901	309

This table shows that the GA indeed performed worse on R_2 than on R_1. However, under the same random seed, the BBDS has the exact performance on R_1 and R_2, indicating that stepping stones do not have any negative impact on the search power of the BBDS. This is because the BBDS essentially takes into account only fitness increase or decrease, not the amount of relative fitness difference. Thus any extra fitness difference contributed by the stepping stones of R_2 does not affect the performance of the BBDS.

Since Royal Road functions are non-deceptive, such landscapes allow BBDS to exhibit the maximum capability to extract information concerning the building blocks whenever they come in sight on the string; thus the global optimum can be reached very quickly. Several empirical results obtained so far indeed show that BBDS significantly outperforms RMHC and traditional GAs on such non-deceptive fitness landscapes.

Although the ISGA needs to employ only a string to attain the optimum of Royal Road functions, this single individual can be fooled by any deceptive schemata. If this is the case, the ISGA with population size one is certainly not enough for attaining gratifying performance. In the next subsection, we present the experimental results of the ISGAs with larger population size on another benchmark test function which bears this fitness landscape feature.

4.2 PERFORMANCE ON 30-BIT BOUNDED DECEPTION PROBLEM

The problems of bounded deception designed by Goldberg et al. (1989) were to investigate the performance of GAs on deceptive functions in which low-order, highly-fit schemata mislead GAs away from global optima and toward the complement of the global optimum. One example of this class is an order-3 fully deceptive function as defined in Table 3.

Table 3: A fully deceptive, order-3 problem

bit	value	bit	value
111	30	100	14
101	0	010	22
110	0	001	26
011	0	000	28

On this 3-bit, deceptive problem, calculations of the average fitness of schema show that GAs are likely to be led toward the complement of the global optimum, i.e., 000, instead of toward the global optimum, 111. To demonstrate the effect of this deception on the search power of GAs, Goldberg et al. (1989) designed a 30-bit deceptive function, **E10**, which is composed of ten consecutive blocks of this 3-bit deceptive function.

In contrast to the non-deceptive feature of Royal Road functions, it is apparent that this 30-bit deceptive function imposes enough difficulty for GAs to arrive at the global maximum (1,1,...,1).

We performed 50 runs of the ISGAs with mate selection schemes A and B on **E10**, based on the second version of BBDS, for population size 40 and 80. The end criterion of the BBDS in this case is the moment that the length of candidate schemata reaches the length of the string. After all the strings reach the end criterion of the BBDS, the independent sampling phase stops and the breeding phase gets started. We then measure the number of function evaluations required to find the global optimum and the results are shown in Table 4 (the standard deviation is given in the parentheses).

Table 4: Experimental Results on **E10**

	Function Evaluations to Optimum	
	Population size 40	Population size 80
Scheme A	11786 (11034)	16794 (12175)
Scheme B	9582 (7889)	11122 (5614)

Notice that the ISGA with mate selection scheme B requires fewer function evaluations than that with scheme A. These results indicate that the ISGA with mate selection B indeed outperforms that with scheme A, which is consistent with the results obtained in (Huang, 2001).

To see how the BBDS version 2 searches this fitness landscape, we show that after the independent sampling phase, only (111) or (000) will emerge at each block, and the probabilities are $\frac{1}{4}$, and $\frac{3}{4}$, respectively (please see Appendix). Thus for a population of 40 individuals, the probability that the population contains no building block (111) at a building-block location is only $(\frac{3}{4})^{40} \approx 1.0 \times 10^{-5}$; and for a population size 80, the probability is $(\frac{3}{4})^{80} \approx 1.0 \times 10^{-10}$. Therefore these two population sizes serve for enough underlying building blocks to construct the global optimum.

To compare total function evaluations used by the two phases in the ISGAs, we show the results in Table 5, where the first element corresponds to the evaluations spent in the independent sampling phase and the second corresponds to that in the breeding phase. In this table, it is clear that scheme B has higher efficiency of exploiting the building blocks found in the independent sampling phase to construct the global optimum.

Table 5: Total Function Evaluations in Two Phases

	Population Size 40	Population Size 80
Scheme A	(1159,10627)	(2322,14472)
Scheme B	(1160,8422)	(2320,8802)

To demonstrate the capability of the ISGAs, we compare their performance (based on population size 40) with that of several different types of GAs: a mGA (Goldberg, Korb, & Deb, 1989), a modified mGA (Goldberg, Deb, Kargupta, & Harik, 1993), a Breeder GA (BGA) (Mühlenbein & Schlierkamp-Voosen, 1993), and two versions of PGA (2pc-wohc, two-point cyclic crossover without hill-climbing, and 2pc-nahc, two-point cyclic crossover with next ascent hill-climbing) (Mühlenbein, 1991). We also ran a simple serial GA over 50 runs (based on a binary tournament selection–with probability 1.0 the fitter of the two randomly sampled individuals is chosen, mutation rate 0.005, two-point crossover rate 0.7, population size 80, and maximum function evaluations 50000 for each run). The experimental results of the ISGAs and other GAs reported can be summarized in Table 6 from which we can see that the ISGAs significantly outperform other GAs.

Table 6: Performance of Several Types of GAs

Mean Function Evaluations to Optimum	
ISGA (Scheme A)	11786
ISGA (Scheme B)	9582
mGA	40600
Modified mGA	26650
BGA	16000
2pc-wohc PGA	21398
2pc-nahc PGA	40500
Serial GA	0 runs reached optimum

5 DISCUSSIONS

One issue that also concerns us is the effect of population size. In Table 4, we see that the ISGA with larger population size has worse performance than with smaller population size. This can be more clearly seen in Table 5. In this table, for the same population size, the function evaluations required in the independent sampling phase for two schemes are almost the same, yet in the breeding phase the difference between two schemes for population size 80 is larger than that for population size 40.

This is the opposite of what has been obtained for simple serial GAs in (Huang, 2001), in which larger population size reduces the performance difference between these two mate selection schemes. So far, the answer for this seeming paradox has not yet been obtained, but we can conjecture that since the ISGAs implement independent sampling of building blocks in the first phase to a maximum degree, they may generate too diverse a population if the population size is large enough. This in turn slows down the evolution of the population in the breeding phase.

How does population diversity affect the searching process for different goals, such as finding a global optimum or forming speciation? From the discussion above, it is clear that larger population size is not always advantageous and we will manage to investigate the relationship between diversity and finding a global optimum in the near future.

6 CONCLUSIONS

In this paper we first present an exploratory method (BBDS) to show how the searching speed of individuals can be improved. Through explicitly acquiring relevant knowledge of candidate building blocks, BBDS outperformed several representative hill-climbing algorithms on non-deceptive Royal Road functions. Then a new class of GAs based on BBDS, i.e., ISGAs, is proposed. In the first phase of ISGAs, implicit parallelism is nicely realized by allowing each individual to accomplish independent building-block sampling to suppress hitchhiking; thus the population is expected to carry diverse promising schemata. Afterwards, with one mate selection scheme that allows individuals to actively choose their mating partners, the efficacy of crossover is enhanced and the ISGAs have been shown to outperform several different GAs on a benchmark test function that is full of deception.

7 FUTURE WORK

Much work remains to be done. The author is now testing the capability of ISGAs on more complicated fitness landscapes, such as the hyperplane defined functions (HDFs) designed by Holland (2000), which parameterize fitness landscapes to encompass features such as hierarchy, poor-linkage, potholes, hills, badlands, ridges, etc. Other research lines are to examine in more detail the impact of mutation and the difference between two-phased and traditional (one-phased) GAs. Afterwards, our hope is to extend the single two-phased procedure to successive two-phased procedures over the course of evolution, i.e., iterating the two phases during the whole run. In addition, other versions of BBDS are worth investigating so that building-block detecting is more effectively implemented. Moreover, a theoretical foundation is needed to explain the whys and wherefores of the excellent performance of ISGAs, and finally our goal is to extend the application of ISGAs to real problems.

Acknowledgments

The author would like to thank Rick Riolo, John Holland for their advice, and Bob Lindsay, Ted Belding, Leeann Fu, and Tom Bersano-Begey for their comments and suggestions.

Appendix

For BBDS version 2, let us start with an initial 3-bit sub-string, for example, at (101) (fitness = 0). Then the first bit is flipped to 0, which causes the fitness to increase to 26; thus this bit's value must be replaced by 0 and its bit-position and new bit-value (i.e., 0) are recorded as the first gene of the candidate schema. After this function evaluation the candidate schema is

(0xx) ("x" represents a not-yet-tested gene) and this sub-string is now (001).

Then under the instruction of step 2, we have to go back to the first bit again and flip it. But since this gene has been already recorded, we do not flip this bit; instead the next bit is temporarily flipped to 1 for test, leading the fitness to decrease to 0. Thus the original value of the sub-string's second bit is kept (i.e., 0) and we record this bit's position and original value as the second gene of the candidate schema, and this candidate schema is now (00x).

Then the BBDS goes to the third bit and flip it. We thus obtain (000), which leads to the fitness of 28. Thus we record this bit's position and new value as the third gene of the candidate schema, which is now (000).

Since the length of this candidate schema reaches the length of this 3-bit sub-string, we stop here.

The process can be symbolized in the following:

$(101) \longrightarrow (0xx) \longrightarrow (00x) \longrightarrow (000).$

Analogously, the collected candidate schemata for the starting points at (100), (110), (010), (001), and (000) will be (000); and those for (011) and (111) will be (111).

We summarize the results as follows:

$(000) \longrightarrow (0xx) \longrightarrow (00x) \longrightarrow (000);$

$(001) \longrightarrow (0xx) \longrightarrow (00x) \longrightarrow (000);$

$(010) \longrightarrow (0xx) \longrightarrow (00x) \longrightarrow (000);$

$(100) \longrightarrow (0xx) \longrightarrow (00x) \longrightarrow (000);$

$(101) \longrightarrow (0xx) \longrightarrow (00x) \longrightarrow (000);$

$(110) \longrightarrow (0xx) \longrightarrow (00x) \longrightarrow (000);$

$(011) \longrightarrow (1xx) \longrightarrow (11x) \longrightarrow (111);$

$(111) \longrightarrow (1xx) \longrightarrow (11x) \longrightarrow (111).$

References

Cantú-Paz, E. (1997). *A survey of parallel genetic algorithms* (IlliGAL Report No. 97003). Urbana, IL: University of Illinois at Urbana-Champaign.

Das, R., & Whitley, L. D. (1991). The only challenging problems are deceptive: Global search by solving order 1 hyperplanes. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 166-173.

Forrest, S., & Mitchell, M. (1993). Relative building block fitness and the building block hypothesis. *Foundations of Genetic Algorithms 2*, 109–126.

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

Goldberg, D. E., Deb, K., & Korb, B. (1991). Don't worry, be messy. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 24-30.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, Accurate optimization of difficult problems using fast messy genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 56-64.

Goldberg, D. E., Korb, B., & Deb, K. (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. (Second edition: MIT Press, 1992.)

Holland, J. H. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation* 8(4):373-391.

Huang, C.-F. (2001). An Analysis of Mate Selection in Genetic Algorithms. *Proc. of 2001 Genetic and Evolutionary Computation Conference (GECCO-2001)*, submitted.

Mitchell, M., Forrest, S., & Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 245–254.

Mitchell, M., Holland, J. H., & Forrest, S. (1994). When will a genetic algorithm outperform hill climbing? *Advances in Neural Information Processing Systems 6*, 51–58.

Mitchell M.(1996). *An introduction to Genetic Algorithms.* Cambridge, MA: MIT Press.

Mühlenbein, H. (1991). Evolution in time and space – The parallel genetic algorithm. *Foundations of Genetic Algorithms*, 316–337.

Mühlenbein, H. (1992). How genetic algorithms really work I: Mutation and hillclimbing. *Proceedings of Parallel Problem Solving from Nature 2*, 15–26.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). *Optimal interaction of mutation and crossover in the breeder genetic algorithm.* Technical Report 93-042, GMD.

Real Royal Road Functions — Where Crossover Provably is Essential

Thomas Jansen
FB 4, LS 2
Univ. Dortmund
44221 Dortmund, Germany
`jansen@ls2.cs.uni-dortmund.de`

Ingo Wegener
FB 4, LS 2
Univ. Dortmund
44221 Dortmund, Germany
`wegener@ls2.cs.uni-dortmund.de`

Abstract

Mutation and crossover are the main search operators of different variants of evolutionary algorithms. Despite the many discussions on the importance of crossover nobody has proved rigorously for some explicitly defined fitness functions $f_n : \{0,1\}^n \to \mathbb{R}$ that a genetic algorithm with crossover can optimize f_n in expected polynomial time while all evolution strategies based only on mutation (and selection) need expected exponential time. Here such functions and proofs are presented for a genetic algorithm without any idealization. For some functions one-point crossover is appropriate while for others uniform crossover is the right choice.

1 INTRODUCTION AND HISTORY

Ideas from biological evolution have influenced the design of systems for various aims, i.e., adaptation, simulation, control, and optimization. Here we consider the optimization, in particular, the maximization of pseudo-boolean functions $f_n : \{0,1\}^n \to \mathbb{R}_0^+$. Evolutionary algorithms use selection operators, search operators, and a stopping criterion. The class of search operators contains mutation where one parent creates one child preferring individuals closer to the parent and crossover where (in most cases) two parents create one or more children which lie in the subcube of $\{0,1\}^n$ spanned by the parents.

There have been long debates which type of search operator is "more important". This paper is not a contribution to this debate. We are interested in specific fitness functions such that crossover is necessary to obtain an evolutionary algorithm (then also called genetic algorithm) where the expected time until an optimal search point is evaluated (called the expected optimization time) is polynomial (instead of exponential).

Holland (1975) has described the possible use of crossover leading to the building-block hypothesis (see also Goldberg (1989)). The well-known schema theorem describes the development of schemata within one step. Hence, in general, it does not lead to results on the expected optimization time. Based on these considerations Mitchell, Forrest, and Holland (1992) have introduced the so-called royal road functions $RR_{n,k} : \{0,1\}^n \to \mathbb{R}_0^+$ (w.l.o.g. $n = mk$) where the set $\{1, \ldots, n\}$ of indices is partitioned to m consecutive blocks of k elements each. Then $RR_{n,k}(x)$ is defined as the number of blocks containing only x_i-bits equal to 1. Mitchell, Holland, and Forrest (1994) have investigated these functions (for an overview see Mitchell (1996)).

It has turned out that mutation-based evolutionary algorithms are quite successful for the royal road functions. The so-called $(1+1)$EA with population size 1 and mutation probability $1/n$ has an expected optimization time of $O(2^k \cdot \frac{n}{k} \log \frac{n}{k})$. We only mention that we can prove that this bound is asymptotically tight. It has been shown in the above mentioned papers that the expected optimization time of an idealized genetic algorithm (IGA) is of order $2^k \cdot \log \frac{n}{k}$. IGA does not consider the negative implications of the hitchhiking effect. Experiments show that the $(1+1)$EA is faster than genetic algorithms on royal road functions. It is also clear that even the idealized GA saves only a polynomial factor of order n/k. Moreover, the analysis of the royal road functions shows that crossover often has simultaneously positive and negative effects and one has to argue carefully to prove that the positive effects are more important.

Watson, Hornby, and Pollack (1998) and Watson and Pollack (1999) have presented another "GA-friendly" fitness function called H-IFF. For $n = 2^k$ we have 2^m natural blocks of length 2^{k-m} each. The "value" of a block is equal to its length and a block is "activated" if all bits in this block have the same value (0 or 1). Finally, H-IFF(x) is the sum of the values of all activated blocks. The interesting aspect of H-IFF is that the blocks "are strongly and non-linearly dependent on one another" (Watson (2000)). Several aspects of this function have been investigated (Watson, Hornby, and Pollack (1998), Watson and Pollack (1999, 2000a, 2000b), Watson (2000)) where part of the analysis is based on methods due to Wright and Zhao (1999). These papers contain many arguments why mutation-based evolutionary algorithms have exponential expected optimization time while genetic algorithms may have polynomial expected optimization time. However, all analytical results have been obtained under some simplifying assumptions. Nevertheless, the discussion on H-IFF has revealed new aspects of crossover.

The focus of our paper is another one. We are interested in upper and lower bounds on the expected optimization time which are proved without any assumption. The aim is to show that genetic algorithms have on some functions polynomial expected optimization time while mutation-based evolutionary algorithms need exponential expected optimization time. The functions are defined just to have the desired properties (as it was the case with RR and H-IFF). Our functions will not have such a clear "schema structure" as RR and H-IFF. However, our aim is to show that we can control for some "GA-friendly" functions all negative aspects of crossover without using an artificial algorithm. The first paper where the use of uniform crossover has been proved rigorously is by Jansen and Wegener (1998). However, they have used an artificial small crossover probability of $1/(n \log^3 n)$ in order to control the hitchhiking effect and, for their example, the expected optimization time for mutation-based algorithms is only super-polynomial, namely of order $n^{\log n}$, and not exponential.

Since one-point crossover is the historically first crossover operator and since one-point crossover was assumed to be adequate for the royal road functions, we first consider this type of crossover operator. In Section 2, we introduce and analyze the so-called real royal road functions for one-point crossover. In Section 3, we do the same for uniform crossover. We finish with some conclusions.

2 REAL ROYAL FUNCTIONS FOR ONE-POINT CROSSOVER

Definition 1. *For $x \in \{0,1\}^n$ let $|x|$ be the number of ones in x, i.e., $x_1 + \cdots + x_n$, and let $b(x)$ be the length of the longest block consisting of ones only, i.e., the largest l such that $x_i = x_{i+1} = \cdots = x_{i+l-1} = 1$ for some i. The real royal road functions for one-point crossover are defined by*

$$R_{n,m}(x) = \begin{cases} 2n^2 & \text{if } x = (1,1,\dots,1) \\ n|x| + b(x) & \text{if } |x| \le n-m \\ 0 & \text{otherwise} \end{cases}$$

We use the notation R_n for the special case $m = \lceil n/3 \rceil$.

The function has the property that, as long as $|x| \le n-m$, the fitness depends on the number of ones and ones which build a block are better than ones that are spread over the vector. The all-ones string is optimal and is surrounded by a large valley of bad points. Also H-IFF has the property that the second-best points are far away from the optimal ones. However, for people who like to see "more smooth" functions we can consider such a variant of the real royal road functions R_n where we assume for the ease of description that n is a multiple of 6:

- If $(2/3)n < |x| \le (5/6)n$, the fitness equals $(10/3)n^2 - 4|x|n + b(x) - |x|$, i.e., the fitness decreases linearly with $|x|$. If $b(x) = |x|$, the fitness decreases from $(2/3)n^2$ to 0.
- If $(5/6)n \le |x| \le n$, the fitness equals $12|x|n - 10n^2 + b(x) - |x|$, i.e., the fitness increases linearly with $|x|$. If $b(x) = |x|$, the fitness increases from 0 to $2n^2$.

This variant is for genetic algorithms even easier than the original function for R_n. The lower bounds for evolution strategies without crossover get a bit worse as we show now.

An evolution strategy starts with an initial population of polynomial size. The individuals are chosen randomly and independently. The probability that such an individual has more than $(2/3)n$ ones is bounded above by $e^{-n/18} = e^{-\Omega(n)}$ (application of Chernoff's bound, see Motwani and Raghavan (1995)). Hence, the probability of having an individual with more than $(2/3)n$ ones is exponentially small. If the evolutionary strategy uses a plus-strategy, i.e., accepts only individuals which are not worse than the given ones, the optimal string has to be produced by mutation from an individual with at most $(2/3)n$ ones.

We allow all mutation probabilities where the bits are flipped independently with the same probability $p \leq 1/2$. Then the probability for producing the all-ones string is maximized for strings with $(2/3)n$ ones. The success probability equals $p^{n/3}(1-p)^{2n/3}$ which is maximized for $p = 1/3$ and therefore exponentially small. This implies that the probability of obtaining the optimum in polynomial time or even in time $2^{\varepsilon n}$ for some small $\varepsilon > 0$ is exponentially small. Evolution strategies may allow to accept individuals with more than $(2/3)n$ ones. Then it is the best to have a blind search without any advice, since the optimal string is a single peak like a needle in the haystack. The search region contains exponentially many points and, therefore, the search takes exponential time. For the smooth variant of the real royal road function it may be sufficient to obtain by mutation a point with more than $(5/6)n$ ones from a point with at most $(2/3)n$ ones. The probability for such an event is exponentially small for all mutation probabilities. If we search within the region of more than $(2/3)n$ and less than $(5/6)n$ ones we even get hints to decrease the number of ones. Moreover, the fraction of strings with at least $(5/6)n$ ones among the set of strings with at least $(2/3)n$ ones is exponentially small. This implies the following result.

Proposition 2. *Evolution strategies (without crossover) need with a probability exponentially close to 1 exponentially many steps to optimize the real royal road function R_n (or its smooth variant).*

We now introduce the steady-state GA (genetic algorithm) which we want to analyze. Steady-state GAs are easier to analyze, since we produce only one new individual per step. We use the parameter $s(n)$ for the population size, the parameter $p_c(n)$ for the probability to apply the operator one-point crossover (the two parents are cut after the ith position, $1 \leq i \leq n-1$ is chosen randomly, and the child takes the first i positions from the first parent and the last $n-i$ positions from the second parent), and the standard choice $p_m(n) = 1/n$ for the probability that bits are flipped during mutation.

Algorithm 3. *Steady-state GA*

1.) *Choose independently and randomly the $s(n)$ individuals of the initial population.*

2.) *With probability $p_c(n)$ go to Step $3'$ and with the remaining probability of $1 - p_c(n)$ go to Step $3''$.*

$3'$.) *Choose two parents x and y from the current population. Let z^* be the result of one-point crossover applied to x and y and let z be the result of mutation applied to z^*.*

$3''$.) *Choose one parent x from the current population. Let z be the result of mutation applied to x.*

4.) *If the fitness of z is smaller than the fitness of the worst individual of the current population, go to Step 2. Otherwise, add z to the population. Let W be the multi-set of individuals in the enlarged population which all have the worst fitness and let W' be the set of those individuals in W which have the largest number of copies in W. Eliminate randomly one element in W' from the current population. Go to Step 2.*

Remark. *For the selection procedure in Step $3'$ and $3''$ we only require that $f(x) \geq f(x')$ implies that the probability of choosing x is at least as large as the probability of choosing x' which implies the same selection probabilities for x and x' if $f(x) = f(x')$.*

Algorithm 3 has no stopping criterion, since we want to estimate the expected optimization time. In order to simplify the control of the well-known hitchhiking effect we have introduced a simple and reasonable rule to enlarge the diversity of the population. Among the worst individuals we eliminate one with the largest number of copies.

Theorem 4. *Let $p_c(n) \leq 1 - \varepsilon$ for some $\varepsilon > 0, m \leq \lceil n/3 \rceil$, and $s(n) \geq m + 1$. Then the expected optimization time of the steady-state GA for the real royal road functions $R_{n,m}$ is bounded above by $O(n \cdot s(n)^2 \cdot \log s(n) + n^2 \cdot s(n) \cdot m + s(n)^2/p_c(n))$. For the typical case where $p_c(n)$ is a positive constant and $s(n) \leq n$ the bound is $O(n^3(m + \log n))$.*

Proof. We consider several phases of the run of the steady-state GA. Each phase has a goal and we estimate the expected time until the goal is reached.

Phase 1. *The goal is that at least one individual has at most $n - m$ or exactly n ones.*

Claim 1. *The expected time for Phase 1 is bounded by $1 + o(1)$.*

Proof. The probability that the initial population has not the desired property equals by Chernoff's bounds $2^{-\Omega(n^2)}$. The probability that mutation produces an individual with the desired properties is much larger than n^{-n} and the expected waiting time for such an event is at most $n^n = 2^{O(n \log n)}$. Including the initial step we have to wait on average $1 + o(1)$ steps. □

Phase 2. *Phase 1 is finished and the goal is that all individuals have exactly $n-m$ ones or we have found the optimum.*

Claim 2. *The expected time for Phase 2 is bounded by $O(n^2 \cdot s(n)/m)$.*

Proof. We pessimistically assume that we do not find the optimum. Increasing the number of ones is for $R_{n,m}$ more important than to increase the length of the largest 1-block. As long as the individuals do not have all $n-m$ ones, we will show that the probability of increasing the number of ones in the population is at least $\varepsilon \cdot m/(e \cdot n)$ leading to a waiting time of $O(n/m)$. This implies the claim, since it is sufficient to produce $s(n)(n-m)$ ones. We still have to prove the lower bound on the probability of increasing the number of ones in the population. With probability at least ε we only perform mutation. If we choose a parent with less than $n-m$ ones, there are at least m 1-bit mutations increasing the number of ones and each has a probability of $\frac{1}{n}(1-\frac{1}{n})^{n-1} \geq 1/(e \cdot n)$. If we choose a parent with exactly $n-m$ ones, there is a probability of $(1-\frac{1}{n})^n \geq 1/e \geq m/(e \cdot n)$ to produce a replica which replaces an individual with less than $n-m$ ones. □

Phase 3. *Phase 2 is finished and the goal is that all individuals x have exactly $n-m$ ones where $b(x) = n-m$ or we have found the optimum.*

Claim 3. *The expected time for Phase 3 is bounded by $O(n \cdot s(n)^2 \cdot \log s(n))$.*

Proof. We pessimistically assume that we do not find the optimum. Then only strings with exactly $n-m$ ones are accepted. Let $b_1 \leq \cdots \leq b_{s(n)}$ be the lengths of the longest 1-blocks of the individuals. Individuals are only replaced with individuals with the same or a larger b-value. Hence, b_1 and $b_1 + \cdots + b_{s(n)}$ are non-decreasing with respect to time. We only consider steps without crossover, since crossover cannot make things worse.

If $b_1 = \cdots = b_{s(n)} = i$, the expected time to obtain an individual with a b-values larger than i is $O(n^2/(n-m-i))$. We may choose any individual. The 1-block with i ones has at least one neighbored 0. There are $n-m-i$ further ones. The 2-bit mutations flipping the neighbored 0 and one of the further $n-m-i$ ones increase the fitness. Each 2-bit mutations has a probability of $(\frac{1}{n})^2(1-\frac{1}{n})^{n-2} \geq 1/(e \cdot n^2)$. The expected waiting time for a good 2-bit mutation is $O(n^2/(n-m-i))$. For each i-value we have to wait once for such an event, $1 \leq i \leq n-m$. Hence, the contribution of such events altogether is $O(n^2 \log n)$.

If $b_1 = i$ and $j > 0$ individuals have a larger b-value, one individual with b-value i is replaced with a better individual if we choose one of the j better individuals and mutation flips no bit. The expected waiting time equals $O(s(n)/j)$. For each of the at most $n-m$ possible i-values we have to consider all values $j \in \{1, \ldots, s(n)-1\}$ leading to the bound $O(n \cdot s(n)^2 \cdot \log s(n))$. Altogether, we have proved Claim 3. □

Phase 4. *Phase 3 is finished and the goal is to obtain a population of individuals containing all possible individuals x with $n-m$ ones and $b(x) = n-m$ at least once or to find the optimum.*

Claim 4. *The expected time for Phase 4 is bounded by $O(n^2 \cdot s(n) \cdot m)$.*

Proof. The number of different second-best optimal strings (i. e., strings x with $|x| = n-m$ and $b(x) = n-m$) equals $m+1$, since the 1-block may start at each of the positions $1, \ldots, m+1$. Here it is essential to have $s(n) \geq m+1$. A second-best individual is of type j if the 1-block starts at position j. If Phase 4 has not been finished, there is some j such that the population contains a type-j individual and no type-$(j-1)$ individual or no type-$(j+1)$ individual. In both cases the probability to choose the type-j individual equals $1/s(n)$, since all individuals have the same fitness. Since crossover cannot have negative effects, we only consider steps without crossover. There is always a 2-bit mutation changing a type-j individual into a type-$(j-1)$ individual (flip the 0 at position $j-1$ and the last one of the block) and also a 2-bit mutation changing a type-j individual into a type-$(j+1)$ individual (flip the first one of the block and the first 0 behind the 1-block). The probability of such a 2-bit mutation is at least $1/(e \cdot n^2)$. In the positive case we obtain a "new" individual with the same fitness as all other individuals. We accept this individual and eliminate one individual which is contained at least twice in the population. Remember that the assumption $s(n) \geq m+1$ ensures such duplicates. Hence, the expected time to increase the number of different individuals is $O(n^2 \cdot s(n))$. The total expected time of this phase is bounded by $O(n^2 \cdot s(n) \cdot m)$, since the number of different individuals has to be increased at most m times. □

Phase 5. *Phase 4 is finished and the goal is to obtain an optimal individual.*

Claim 5. *The expected time for Phase 5 is bounded by $O(s(n)^2/p_c(n))$.*

Proof. Because of the selection scheme we always have at least one type-1 individual $1^{n-m}0^m$ and one type-$(m+1)$ individual 0^m1^{n-m}. The probability of choosing a crossover step with this pair of individuals is at least $p_c(n)/s(n)^2$. Each crossover position p where $m \leq p \leq n-m$ leads to the child 1^n. The probability of such a position is $\frac{n-2m+1}{n-1} \geq \frac{1}{3}$. Moreover, the probability that mutation does not destroy 1^n is at least $1/e$. Altogether, the waiting time for such a good step is $O(s(n)^2/p_c(n))$. □

The theorem follows by summing up the expected times for all phases. □

We see that 1-blocks are building blocks. However, only 1-blocks at the beginning or at the end of the string are useful to obtain the optimum by 1-point crossover.

We have introduced $R_{n,m}$ as real royal road functions for one-point crossover, since we obtain for $m = \lceil n/3 \rceil$ a trade-off of exponential time for evolution strategies without crossover and polynomial time for our steady-state GA.

We also investigate the steady-state GA where one-point crossover is replaced with uniform crossover (for all positions i independently choose x_i or y_i with probability 1/2). The analysis of the first four phases can be used without changes. The probability that uniform crossover creates 1^n from $1^{n-m}0^m$ and 0^m1^{n-m} equals 2^{-2m}. This is polynomially bounded only if $m = O(\log n)$. If $m = \lfloor \log n \rfloor$, we still get the bound $O(n^4)$ for the expected optimization time (if $p_c(n)$ is a constant and $s(n) \leq n$). However, evolution strategies (with a single individual) need time $\Theta(n^{\lfloor \log n \rfloor})$ in this situation. Hence, we obtain the same trade-off as Jansen and Wegener (1999), but for non-artificial values of $p_c(n)$.

3 REAL ROYAL ROAD FUNCTIONS FOR UNIFORM CROSSOVER

Real royal road functions for uniform crossover are harder to design. The reason is that 1-point crossover can only create $n-1$ different children. For uniform crossover of x and y we have two possibilities. If the Hamming distance between x and y is small, also the number of different possible children is small. However, in this situation also mutation can create these children with not too small probability. If the Hamming distance between x and y is large, each possible child has a vanishing probability to be created.

In order to simplify the description we assume that $n = 2m$ and $m = 3k$. The input $x \in \{0,1\}^n$ is described as pair $x = (x', x'')$ where x' and x'' both have length m. Furthermore, $x'' = (x''_1, x''_2, x''_3)$ where x''_1, x''_2, and x''_3 all have length k. We say that $x'' \in C$ (C is a circle) if $x'' \in \{0^i1^{m-i}, 1^i0^{m-i} \mid 0 \leq i \leq m-1\}$. The circle is a closed path (Hamming distance 1 between neighbored points) of length $2m = n$. We say that $x'' \in T$ (T is the target) if each of the substrings x''_1, x''_2, and x''_3 contains $\lfloor k/2 \rfloor$ ones and $\lceil k/2 \rceil$ zeros. For strings a and b let $H(a,b)$ be the Hamming distance between a and b. For a set of strings B let $H(a,B)$ be the smallest Hamming distance between a and some $b \in B$.

Definition 5. *The real royal road functions for uniform crossover are defined by*

$$R^*_n(x', x'') = \begin{cases} n - H(x'', C) & \text{if } x' \neq 0^m \\ & \text{and } x'' \notin C \\ 2n - H(x', 0^m) & \text{if } x'' \in C \\ 0 & \text{if } x' = 0^m \\ & \text{and } x'' \notin C \cup T \\ 3n & \text{if } x' = 0^m \\ & \text{and } x'' \in T \end{cases}$$

This definition needs some explanation. With overwhelming probability, the initial population contains only individuals where x' is far from 0^m. Then the fitness function gives advice that x'' should be changed into a "circle string". This can be done efficiently with mutations only. It is unlikely to create in this phase a string where $x' = 0^m$. If $x'' \in C$, the fitness increases with decreasing distance of x' to 0^m. Then we will have individuals where $x' = 0^m$ and $x'' \in C$. The steady-state GA will ensure that the population will contain all possible $x'' \in C$ (if the population is large enough). However, we are far from the optimal strings where $x' = 0^m$ and $x'' \in T$. Uniform crossover of $0^m0^i1^{m-i}$ and $0^m1^i0^{m-i}$ has a good chance to create an optimal string and mutation only cannot do this job efficiently.

We admit that this function is an artificial one, but it is the first one where one can prove that uniform crossover decreases the expected optimization time from exponential to polynomial. As for the real royal road functions for 1-point crossover it is possible to define a "smooth" variant of R^*_n. We omit this technical definition.

The analysis of evolution strategies follows the lines of the corresponding analysis in Section 2. The probability that the initial population of polynomial size contains an individual x where x' has less than $m/3$ ones is exponentially small. As long as $x' \neq 0^m$ and $x'' \notin C$

the search on the first half, namely x', is a search for the needle 0^m in a haystack. Hence, the probability of finding in polynomial time a string where $x' = 0^m$ and $x'' \notin C$ is exponentially small. Afterwards, $x'' \in C$ is better than $x'' \notin C$ with the only exception of $x' = 0^m$ and $x'' \in T$. With small mutation probabilities like $1/n$ it is easy to find strings x where $x' = 0^m$ and $x'' \in C$. With large mutation probabilities we miss the strings where $x' = 0^m$. Hence, we need a mutation from x where $x' = 0^m$ and $x'' \in C$ to some y where $y' = 0^m$ and $y'' \in T$. The minimal Hamming distance between some $x'' \in C$ and some $y'' \in T$ is $\Omega(n)$. This follows easily, since two of the three strings x''_1, x''_2, and x''_3 are of type 0^k or 1^k. Hence, we need a mutation step where none of the first m bits flips and a constant fraction of the last m bits flips. The last event has an exponentially small probability if the mutation probability decreases with n. For larger mutation probabilities the first event has an exponentially small probability. This implies the following result.

Proposition 6. *Evolution strategies (without crossover) need with a probability exponentially close to 1 exponentially many steps to optimize the real royal road function R_n^* (or its smooth variant).*

Theorem 7. *Let $p_c(n)$ be some positive constant smaller than 1 and $s(n) \geq n$. Then the expected optimization time of the steady-state GA for the real royal road function R_n^* for uniform crossover is bounded above by $O(n^2 \cdot s(n))$ which is $O(n^3)$ if $s(n) = O(n)$.*

Proof. We follow the same proof strategy as in the proof of Theorem 4.

Phase 1. *The goal is that $x'' \in C$ for all individuals $x = (x', x'')$ of the population or we have found the optimum.*

Claim 1. *The expected time for Phase 1 is bounded by $O(n^2 \cdot s(n))$.*

Proof. We pessimistically assume that we do not find the optimum. As long as there is an individual $x = (x', x'')$ where $x' = 0^m$ and $x'' \notin C \cup T$ (these are the only strings with fitness 0), one of them is eliminated with a positive constant probability. We only consider steps without crossover. Either we choose one of the described individuals. Then it is sufficient that at least one of the first $m = n/2$ bits flips. Otherwise, it is sufficient to construct a replica. Hence, on the average, after $O(s(n))$ steps we have no individual with fitness 0.

Afterwards, we like to eliminate the individuals where $x' \neq 0^m$ and $x'' \notin C$. The "distance" of the population to our goal is measured as the sum of all $H(x'', C)$ where $x = (x', x'')$ belongs to the population, $x' \neq 0^m$, and $x'' \notin C$. This distance is smaller than $s(n) \cdot m$ and the goal is to decrease it to 0. Because of our selection procedure the distance cannot increase. Therefore, it is sufficient to consider steps without crossover. If we select an individual x, where $x' \neq 0^m$ and $x'' \notin C$, for mutation, there is at least one 1-bit mutation which creates an individual z where $H(z'', C) < H(x'', C)$. If we select an individual where $x'' \in C$, the distance of the population decreases if mutation creates a replica. Hence, the expected waiting time to decrease the distance is $O(n)$ (this is the waiting time for a special 1-bit mutation). We have to wait for such an event for at most $s(n) \cdot m$ times which proves Claim 1. (The bound of Claim 1 can be improved, since often there are many good 1-bit mutations. However, this will not improve the bound of the theorem.) ☐

Phase 2. *Phase 1 is finished and the goal is that $x' = 0^m$ and $x'' \in C$ for all individuals $x = (x', x'')$ of the population or we have found the optimum.*

Claim 2. *The expected time for Phase 2 is bounded by $O(n^2 \cdot s(n))$.*

Proof. We pessimistically assume that we do not find the optimum. Hence, we only have to consider individuals $x = (x', x'')$ where $x'' \in C$. Now the "distance" of the population to the goal is measured as the sum of all $H(x', 0^m)$ where $x = (x', x'')$ belongs to the population. The distance is at most $s(n) \cdot m$ and the goal is to decrease it to 0. The situation is similar to the proof of Claim 1. The distance does not increase and we consider only steps without crossover. If we select an individual x where $x' \neq 0^m$ for mutation, there is at least one 1-bit mutation which creates z where $H(z', 0^m) < H(x', 0^m)$. Otherwise, $x' = 0^m$. If mutation creates a replica and the distance of the population is positive, we decrease the distance. Altogether, we have proved Claim 2 (and also the bound in Claim 2 can be improved). ☐

Phase 3. *Phase 2 is finished and the goal is to obtain a population containing all possible individuals x where $x' = 0^m$ and $x'' \in C$ at least once or to find the optimum.*

Claim 3. *The expected time for Phase 3 is bounded by $O(n^2 \cdot s(n))$.*

Proof. We pessimistically assume that we do not find the optimum. Then the population only contains individuals x where $x' = 0^m$ and $x'' \in C$. The circle C is a closed path where each point has two neighbors with Hamming distance 1. As long as the goal is not reached, the population contains at least two individuals, say $x = (x', x'')$ and $y = (y', y'')$, such that $x' = y' = 0^m, x'', y'' \in C$, and both individuals have a Hamming neighbor, say $\tilde{x} = (\tilde{x}', \tilde{x}'')$ and $\tilde{y} = (\tilde{y}', \tilde{y}'')$ resp., such that $\tilde{x}' = \tilde{y}' = 0^m, \tilde{x}'', \tilde{y}'' \in C$, and $\tilde{x}$ and $\tilde{y}$ do not belong to the current population.

We define the "distance" of the current population to the goal as the number of individuals $z = (z', z''), z' = 0^m$, and $z'' \in C$, which are not contained in the population. The distance is at most $n - 1$ and the goal is to decrease it to 0. Our selection procedure implies that the distance cannot increase. Hence, we look for the expected time to decrease the distance. This happens if we choose a step without crossover, select one of the individuals described above, and perform the "good" 1-bit mutation. Here we need the assumption that $s(n) \geq n = |C|$. Hence, the expected waiting time to decrease the distance is bounded by $O(n \cdot s(n))$ which proves Claim 3. □

Phase 4. *Phase 3 is finished and the goal is to obtain an optimal individual.*

Claim 4. *The expected time for Phase 4 is bounded by* $O(n^{3/2} \cdot s(n))$.

Proof. Because of our selection procedure we always have all individuals $x = (x', x''), x' = 0^m, x'' \in C$, in our population if we have not found the optimum. Here we only consider steps with crossover. Let $x = (x', x'')$ be the first chosen parent. Then the probability of choosing $y = (y', y'')$ where $y' = 0^m$ and $y''_i = 1 - x''_i$ for all i is at least $1/s(n)$. The reason is that y is contained in the population and that all individuals of the population have the same fitness and, therefore, the same chance of being chosen. Let $\tilde{z}$ be the result of uniform crossover applied to x and y. Then $\tilde{z}' = 0^m$ and $\tilde{z}''$ is a random string, since x'' and y'' have different bits at all positions. We have $\tilde{z}'' = (\tilde{z}''_1, \tilde{z}''_2, \tilde{z}''_3)$. The probability that $\tilde{z}''_j, 1 \leq j \leq 3$, contains exactly $\lfloor k/2 \rfloor$ ones and $\lceil k/2 \rceil$ zeros is $\Theta(k^{-1/2})$ (the usual estimate of $\binom{k}{\lceil k/2 \rceil} 2^{-k}$ by Stirling's formula). Hence, the probability that $\tilde{z}'' \in T$ is $\Theta(k^{-3/2})$. Finally, there is a probability of at least $1/e$ that mutation does not destroy $\tilde{z}$. Hence, the success probability is at least $\Omega(s(n)^{-1} \cdot k^{-3/2}) = \Omega(s(n)^{-1} \cdot n^{-3/2})$ and the expected waiting time for a success is bounded by $O(n^{3/2} \cdot s(n))$. □

The theorem follows by summing up the expected times for all phases. □

4 CONCLUSIONS

We have presented for the first time functions where it can be proved without any assumption that evolution strategies without crossover need with overwhelming probability exponential time to find the optimum while a realistic steady-state GA has a polynomial expected optimization time. One-point crossover is successful for a function with building blocks. However, the example function has the property that only two of the building blocks are useful to create the optimum by crossover. The real royal road function where uniform crossover works has no building blocks. Here it is essential that the population contains quite different individuals and that it is possible to create individuals "in the middle of the population".

Nevertheless, the real royal road functions are very special. The results heer can be seen as a first step of analyzing crossover rigorously. The next steps are to obtain results for other fitness functions and results about the diversity of populations. It will take many major steps to prove rigorously that crossover is essential for typical applications.

References

Forrest, S., and Mitchell, M. (1993). Relative building-block fitness and the building-block hypothesis. 2. Workshop Foundations of Genetic Algorithms (FOGA), Morgan Kaufmann, San Mateo, Calif.

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

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The Univ. of Michigan Press, Ann Arbor, Mich.

Jansen, T., and Wegener, I. (1999). On the analysis of evolutionary algorithms – a proof that crossover really can help. 7. Europ. Symp. on Algorithms (ESA), LNCS 1642, 184-193, Springer, Berlin.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms.* Chapter 4.2. MIT Press, Cambridge, Mass.

Mitchell, M., Forrest, S., and Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. In F. J. Varela and P. Bourgine (Eds.): *Toward a Practice of Autonomous Systems.* Proc. of the First European Conf. on Artificial Life, MIT Press, Cambridge, Mass.

Mitchell, M., Holland J. H., and Forrest, S. (1994). When will a genetic algorithm outperform hill-climbing? Advances in NIPS 6, Morgan Kaufmann, San Mateo. Calif.

Motwani, R., and Raghavan, P. (1995). *Randomized Algorithms.* Cambridge Univ. Press, Cambridge.

Watson, R. A. (2000). Analysis of recombinative algorithms on a non-separable building-block problem. 5. Workshop Foundations of Genetic Algorithms (FOGA), to appear.

Watson, R. A., Hornby, G. S., and Pollack, J. B. (1998). Modeling building-block interdependency. 5. Symp. Parallel Problem Solving from Nature (PPSN), LNCS 1998, 97-106, Springer, Berlin.

Watson, R. A., and Pollack, J. B. (1999). Hierarchically-consistent test problems for genetic algorithms. Congress on Evolutionary Computation (CEC), 1406-1413, IEEE Press.

Watson, R. A., and Pollack, J. B. (2000a). Symbiotic combination as an alternative to sexual recombination in genetic algorithms. 6. Symp. Parallel Problem Solving from Nature (PPSN). LNCS 1917, 425-434, Springer, Berlin.

Watson, R. A., and Pollack, J. B. (2000b). Recombination without respect: schema disruption in genetic algroithm crossover. Proc. of the Genetic and Evolutionary Computation Conference (GECCO), 112-119, Morgan Kaufmann, San Mateo, Calif.

Wright, A. H., and Zhao, Y. (1999). Markov chain models of genetic algorithms. Proc. of the Genetic and Evolutionary Computation Conference (GECCO), 734-741, Morgan Kaufmann, SanMateo, Calif.

A Genetic Algorithm Architecture by Coordinating Exploration and Exploitation

Rui JIANG† K.Y. SZETO‡ Yupin LUO† Dongcheng HU†

† : *Department of Automation*, *Tsinghua University*,
Beijing 100084, *P.R. China*
Email: {rjiang, luo}@mail.au.tsinghua.edu.cn

‡ : *Department of Physics*,
Hong Kong University of Science and Technology,
Clear Water Bay Kowloon Hong Kong, *P.R. China*
Email: phszeto@ust.hk

Abstract

We introduce a new genetic algorithm architecture based on the careful balance of exploration and exploitation in the solution space of a typical problem in optimization. In this architecture, the population in each generation consists of three sub-populations: a preserved part, a reproduced part and a randomized part. Two parameters are introduced to control the percentage of each sub-population efficiently to achieve good balance between the processes of exploration and exploitation while doing optimization. By modeling the algorithm as a homogeneous finite Markov chain, the new genetic algorithm is shown to converge towards global optimum of the problem at hand. Experiments are designed to test the algorithm using Rastrigin function, Griewangk function and Schaffer function. Data analysis using average success ratio, average objective calculating number, average first passage time to solution, and standard deviation of first passage time are made and compared with canonical genetic algorithm, elitist genetic algorithm, and steady genetic algorithm. The results show strong evidence that our algorithm is superior in performance in terms of economy, robustness and efficiency.

1 INTRODUCTION

Genetic algorithms (GA) are versatile evolutionary computation techniques that are largely based on the principle of survival of the fittest (Holland,1975). Through simple encoding schemes of the representation of individuals in a population of potential solutions, complex phenomena are described by their collective evolution. Without making assumptions on continuity, existence of derivatives, uni-modality and other matters, the Darwinian selection mechanism drives the population in a parallel manner in the exploration of the solution space, while the natural selection of fit individuals to survive and reproduce provide the necessary exploitation of the knowledge built from past generations (Goldberg,1989) (Pan,1998). These attractive features of genetic algorithms have been exploited in many applications in different areas of science and engineering, such as forecasting, machine learning, image processing, and pattern recognition (Szeto,1998) (Szeto,1999) (Hu,1999) (Ankerbrandt,1990).

Conceptually, there exist two different kinds of operation in genetic algorithms: *exploration* and *exploitation*. The common genetic operators such as crossover and mutation are mechanisms for exploration of the solution space (Pan,1998). In crossover, existing genes are recombined to obtain chromosomes which are more fit in the immediate environment. In mutation, new genes are brought into the population to maintain the diversity of chromosomes. Both operators aim at exploring different regions of the solution space. In contrast, selection is an operator that takes an adequate exploitation to obtain useful information in the current population by guiding the algorithm towards search areas that are more probable to contain the global optimum. Exactly how useful information can be exploited by the selection mechanism is one of the subtle research questions, but the aim to get quickly to solution with high reliability is the common goal in application. In this point of view, it is easy to understand that the pivotal problem in GA is to find an intelligent way to coordinate exploration and exploitation, without sacrificing the efficiency, while making sure that the method used is sufficiently general and not specific to a particular class of problems. We are here addressing this fundamental problem by formulating a new genetic algorithm, in a sufficiently general framework with clarity so that applications can be easily implemented in different problems.

A prevalent method in GA is to assign survival probabilities to corresponding individuals and tune the probabilities to obtain the balance between exploration and exploitation (Goldberg,1989)(Pan,1998). However, the process of selecting parameters is itself subjective, in the sense that one should refer to the nature of the given problem and characteristic features of the solution space. Indeed, experiential strategies and intelligent adaptation of parameters are not possible or difficult to be implemented in this common application of GA. Adaptive schemes that adjust the probabilities of crossover and mutation while running the GA have also been considered to address this problem (Srinivas,1994). Nevertheless, adjustment between exploration and exploitation in this manner is not efficient, as both crossover and mutation are operators for exploration. On the other hand, we can also divide the population into sub-populations with sensible criteria and attach meaning to the divided population. We will address this issue later in the paper. For now, we should make two important observations. (1) Since the way that the population is divided into sub-population does not depend on the details of the genetic operators used in other parts of the algorithm, we have a general scheme in tuning the relative levels of exploration and exploitation by controlling the size of each sub-population. (2) Since the architecture and the selection strategy do not depend on the specific features of the problem, the encoding scheme, and various genetic operators, the parameter control on the selection mechanism is in principle very general. These two observations suggest that a simple philosophy underlying the new genetic algorithm is to find a general parameter control in selection strategies that enable efficient tuning of the importance of exploration and exploitation during evolution by controlling the sizes of the sub-populations.

We now introduce the new genetic algorithm by outline the criteria and meaning in the division of populations in the architecture. The newly generated population consists of three sub-populations: a preserved part, a reproduced part and a randomized part. The amount of preserved individuals represents the relative importance of exploration and exploitation in the genetic algorithm. The amount of reproduced individuals stands for the effect of various genetic operators while the algorithm explores the solution space. The amount of randomized individuals adjusts the efficiency of exploration in a slight way, so that the directed search is balanced by the injection of new individuals that can come from a region possibly far from the searched area. This method of control on exploration and exploitation leads to efficient genetic algorithms that fulfill optimization tasks with flexibility, robustness, and speed.

To ensure that our new architecture of genetic algorithm does work generally, we must address the problem of convergence. Here we provide several theorems on convergence that makes use of the insights in previous works (Rudolph,1994)(Dinabandhu,1996)(Suzuki,1995). In order to address the problem that simple genetic algorithm cannot guarantee global convergence despite its ergodicity (Rudolph,1994), various elitist genetic algorithms have been proposed (Rudolph,1994), (Dinabandhu,1996) and (Suzuki,1995). To analyze their convergence properties, the population in each generation defines a distinct state of a finite Markov chain. Using the theories about finite Markov chains (Iosifescu,1980), we succeed in proving the convergence of our new genetic algorithm.

We introduce the algorithm architecture in section 2, and discuss the convergence properties in section 3. In section 4, dynamic properties of the algorithm are discussed in the context of several evaluation criteria. In section 5, experiments designed for testing the algorithm are described and the results are analyzed and compared with several typical GAs. Finally, in section 6, conclusions are presented.

2 ALGORITHM ARCHITECTURE

We only discuss genetic algorithms with a fixed number N of individuals in population. Given an initialized population $P(0)$, genetic operators will be applied on it to generate population $P(1)$, thereby yielding a series of populations $\mathbf{P}=\{P(0),P(1),\ldots,P(t),P(t+1),\ldots\}$. If the individual corresponding to the global optimum appears in certain population in the series, we say that the global optimum is found. In practice, a terminating criterion should be used to stop the algorithm to obtain a finite part of the infinite series, $\mathbf{P}(t)=\{P(0),P(1),\ldots,P(t)\}$. The basic problem of this iterating procedure is to get $P(t+1)$, the population in time $t+1$, from $P(t)$, the population in time t. In a GA with our architecture, $P(t+1)$ is constructed by three sub-populations: a preserved part $P_1(t)$, a reproduced part $P_2(t)$ and a random generated part $P_3(t)$. Corresponding parameters, r_1, r_2 and r_3 are introduced to the architecture to control the amount of each sub-population. In each generation, individuals are sorted according to their fitness and some fit ones are copied to form $P_1(t)$. Then, certain strategy is employed to select some individuals out of the sorted ones. Genetic operators are then performed on them to generate $P_2(t)$. Finally, some individuals are generated randomly to form $P_3(t)$. Fig.1 summarizes this procedure.

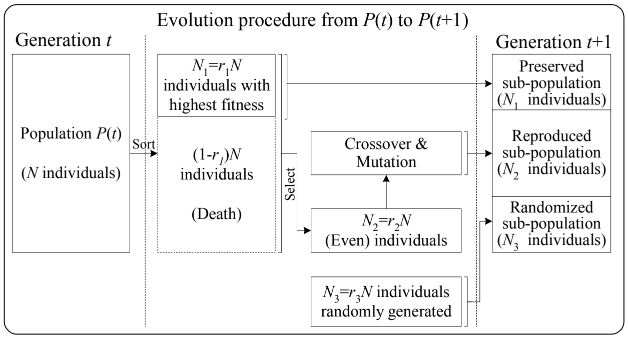

Evolution procedure from P(t) to P(t+1)
Generation t
Generation t+1
Population P(t)
(N individuals)
Sort
N1=r1N individuals with highest fitness
(1-r1)N individuals
(Death)
Select
Crossover & Mutation
N2=r2N (Even) individuals
N3=r3N individuals randomly generated
Preserved sub-population (N1 individuals)
Reproduced sub-population (N2 individuals)
Randomized sub-population (N3 individuals)

Fig.1: The algorithm architecture

Parameters r_1, r_2 and r_3 have the following meaning:

- r_1: *Preservation fraction*. It determines the number of individuals $N_1= r_1N$ in $P(t+1)$ that are directly copied from $P(t)$. These individuals form $P_1(t)$.
- r_2: *Reproduction fraction*. It determines the number of individuals $N_2= r_2N$ in $P(t+1)$ that are generated by genetic operators. These individuals form $P_2(t)$.
- r_3: *Random fraction*. It determines the number of individuals $N_3=r_3N$ in $P(t+1)$ that are generated randomly. These individuals form $P_3(t)$.

Since the total number of individuals, $N_1+N_2+N_3=N$, is fixed, $r_1+r_2+r_3=1$. This means that only two parameters r_1 and r_2 are independent. Note that $N_1=r_1N>0$ and $N_2=r_2N>0$ in order to have the possibility of global convergence. Now the problem is to make a connection between these two parameters and the balance of exploration and exploitation by controlling the sizes of these sub-populations. To achieve this aim, we must first address two central questions concerning the operation procedures in defining the architecture. (1) *How do we determine the preserved individuals?* (2) *How do we select individuals for reproduction?* According to analyses on existing selection strategies (Pan,1998), we can use the ranks of the fitness of individuals for question (1) and the roulette wheel selection based on ranks for question (2). We now define the procedures of evolution for our new genetic algorithm by the following steps:

Step 1: Calculate the preservation fraction r_1, the reproduction fraction r_2, and the random fraction r_3. Calculate N_1, N_2, and N_3, the amount of each sub-population.
Step 2: Sort individuals in population $P(t)$ according to their fitness to form a descending series.
Step 3: Select anterior N_1 individuals from the series to form the preserved population $P_1(t)$.
Step 4: Select N_2 individuals from the series with roulette wheel strategy. Perform crossover and mutation on them. Evaluate them to form the reproduced population $P_2(t)$.
Step 5: Generate N_3 individuals randomly and evaluate them to form the random population $P_3(t)$.
Step 6: Generate the population $P(t+1)$ by letting $P(t+1) = P_1(t) \cup P_2(t) \cup P_3(t)$.

3 CONVERGENCE PROPERTIES

Without loss of generality, we assume a binary encoding scheme in the problem of finding the maximum of a function $f(x)$. Here x is defined on a discrete space that can be spanned by binary strings. An individual is defined as a string with length L over the finite set $\mathbb{B} = \{0,1\}$. There are 2^L distinct individuals over $\mathbb{B}$, forming the collection $\mathbf{I}$. A function $fit(\bullet)$ defined on $\mathbf{I}$ is introduced to evaluate individuals and $fit(I) > 0, \forall I \in \mathbf{I}$. Let $\mathbf{F}$ be the collection that contains all the possible fitness values, i.e., $\mathbf{F} = \{F : F = fit(I)$, any $I \in \mathbf{I}\}$. Let b be the number of elements in $\mathbf{F}$. Note that $b \le 2^L$. According to the fitness of each individual, the collection $\mathbf{I}$ can be divided to non-empty sub-sets $\{\mathbf{I}_i\}$, $\mathbf{I}_i = \{I : I \in \mathbf{I}$ and $fit(I) = F_i\}$; i=1,2,…,b. Apparently, $\mathbf{I}_i \ne \phi$, i=1,2,…,b, $\mathbf{I}_i \cap \mathbf{I}_j = \phi$, $\forall i \ne j$, and $\bigcup_{i=1}^{b} \mathbf{I}_i = \mathbf{I}$. We can sort the collection $\mathbf{F} = \{F_1, F_2, ..., F_b\}$ so that $F_1 > F_2 >, …, > F_b$. After this sorting, we see that F_1 is the global optimum fitness F^* and $\mathbf{I}_1$ contains all the individuals whose fitness equal to F^*.

A population P is a collection of N individuals. Let I_k (k=1,2,…,N) be the k-th individual in P and denote P as $\{I_1, I_2, …, I_N\}$. Since individuals in the population are not required to be unique, we can compute the possible number M of distinct populations defined on the collection $\mathbb{B}$ using theorems in combinatorics and get

$$M = \begin{pmatrix} 2^L + N - 1 \\ N \end{pmatrix}$$

All possible populations form a collection $\mathbf{P}$ with M elements. A fitness-evaluating function, defined to be $fit(P)$=max$\{fit(I)\}$, $\forall I \in P$, is introduced to measure the maximum fitness of a given population P. Since the collection of population fitness is identical with the collection of individual fitness, we have $F_b \le fit(P) \le F_1$, $\forall P \in \mathbf{P}$. According to their fitness, $\mathbf{P}$ can be divided to non-empty sub-sets $\{\mathbf{P}_i\}$, with $\mathbf{P}_i$ being itself a collection of population: $\mathbf{P}_i = \{P : P \in \mathbf{P}$ and $fit(P) = F_i\}$, i=1,2,…,b. Note that $\mathbf{P}_i \ne \phi$, i=1,2,…,b; $\mathbf{P}_i \cap \mathbf{P}_j = \phi$, $\forall i \ne j$; $\bigcup_{i=1}^{b} \mathbf{P}_i = \mathbf{P}$; and $\mathbf{P}_1$ contains all those populations with fitness F^*. Let p_i be the number of elements in $\mathbf{P}_i$. (Note $\sum_{i=1}^{b} p_i = M$). Let P_{ij} be the j-th element in set $\mathbf{P}_i$, j=1,2,…,p_i, i=1,2,…,b. With the operation of genetic operators, the population P_{ij} may directly transit to population P_{kl} and we denote this state transition by $P_{ij} \to P_{kl}$, with transition probability $\Pr(P_{ij} \to P_{kl}) = p_{ij,kl}$. Similarly we can define the transition directly from the population P_{ij} to *any* population in collection $\mathbf{P}_k$, and denote this population transition by $P_{ij} \to \mathbf{P}_k$, with transition probability $\Pr(P_{ij} \to \mathbf{P}_k) = p_{ij,k}$. Finally, the transition directly from the collection $\mathbf{P}_i$ to the collection $\mathbf{P}_k$ is denoted by $\mathbf{P}_i \to \mathbf{P}_k$ with transition probability $\Pr(\mathbf{P}_i \to \mathbf{P}_k) = p_{i,k}$. Note that we have the following general results on transition probabilities:

$$p_{ij,k} = \sum_{l=1}^{p_k} p_{ij,kl}\ , \sum_{k=1}^{b} p_{ij,k} = 1 \ \text{ and } \ p_{i,k} \ge p_{ij,k}$$

Here j=1,2,…,p_i, i=1,2,…,b; l=1,2,…,p_k, and k=1,2,…,b.

With these definitions, we now define global convergence and prove that genetic algorithms with our architecture converge to the global optimum.

Definition 1: Let $P(n)$ be the population in the n-th generation of a given genetic algorithm and F^* the global optimum fitness. We say that the algorithm has the property of global convergence when

$$\lim_{n\to\infty} \Pr\{fit[P(n)] = F^*\} = 1 \qquad (1)$$

Since F^*=F_1, global convergence also means

$$\lim_{n\to\infty} \Pr\{P(n) \in \mathbf{P}_1\} = 1 \qquad (2)$$

Lemma 1 (Iosifescu,1980): Let $\mathbf{P}$ be an $n \times n$ reducible stochastic matrix, that is, it can be brought into

$$\mathbf{P} = \begin{pmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{R} & \mathbf{T} \end{pmatrix}$$

by applying the same permutations to rows and columns. Here $\mathbf{C}$ is an $m \times m$ primitive stochastic matrix and $\mathbf{R}, \mathbf{T} \ne \mathbf{0}$. Then

$$\mathbf{P}^\infty = \lim_{k\to\infty} \mathbf{P}^k = \lim_{k\to\infty} \begin{pmatrix} \mathbf{C}^k & \mathbf{0} \\ \sum_{i=0}^{k-1} \mathbf{T}^i \mathbf{R} \mathbf{C}^{k-i} & \mathbf{T}^k \end{pmatrix} = \begin{pmatrix} \mathbf{C}^\infty & \mathbf{0} \\ \mathbf{R}^\infty & \mathbf{0} \end{pmatrix}$$

is a stable stochastic matrix with $\mathbf{P}^\infty = \mathbf{e}' \cdot \mathbf{p}^\infty$, where $\mathbf{e} = (1,1,\cdots,1)$, $\mathbf{p}^\infty = \mathbf{p}^0 \cdot \lim_{k\to\infty} \mathbf{P}^k = \mathbf{p}^0 \cdot \mathbf{P}^\infty$ is unique regardless of the initial distribution $\mathbf{p}^0$, and $\mathbf{p}^\infty$ satisfies: $p_i^\infty > 0$ for $1 \le i \le m$ and $p_i^\infty = 0$ for $m < i \le n$.

Theorem 1: For a genetic algorithm with the basic architecture with $N_1 = r_1 \cdot N > 0$ and $N_2 = r_2 \cdot N > 0$, we have for all j=1,2,…,p_i, i=1,2,…,b, and k=1,2,…,b,

$$p_{ij,k} \begin{cases} > 0, & k \le i \\ = 0, & k > i \end{cases}$$

Proof: Follow the method of Dinbandhu et al (Dinabandhu,1996) pp 739. ■

Theorem 2: For a genetic algorithm with our architecture with $N_1 = r_1 \cdot N > 0$ and $N_2 = r_2 \cdot N > 0$, we have for all i=1,2,…,b, and k=1,2,…,b,

$$p_{i,k} \begin{cases} > 0, & k \le i \\ = 0, & k > i \end{cases}$$

Proof: Theorem 2 can be obtained from Theorem 1. Note that $p_{i,k} \ge p_{ij,k}$, $j = 1,2,...,p_i$, $i = 1,2,...,b$, $k = 1,2,...,b$. ■

Theorem 1 tells us that a population may transit to another one with equal or higher fitness but cannot transit to a population with lower fitness, while theorem 2 tells us that when the population of an algorithm belongs to a collection of populations with certain fitness, it can transit to a collection of populations with equal or higher fitness. Note that we do not have the reverse result.

Theorem 3: A genetic algorithm with our architecture can guarantee the convergence to the global optimum.

Proof: We prove this by showing that (2) in our definition of global convergence can be satisfied by our new genetic algorithm. First note that each $\mathbf{P}_i$, i=1,2,…,b can be treated as a state in a homogenous finite Markov chain. Let η_i be the probability that the algorithm stays in $\mathbf{P}_i$. Note that $\eta_i > 0$ for i=1,2,…,b and $\sum_{i=1}^{b} \eta_i = 1$. Since the transition probability $p_{i,k}$ from a state $\mathbf{P}_i$ to another state $\mathbf{P}_k$ satisfies $p_{i,k} > 0$, $\forall i \ge k$ and $p_{i,k} = 0$, $\forall i < k$. The transition matrix of this finite Markov chain is therefore given by

$$\mathbf{P} = \begin{pmatrix} p_{1,1} & 0 & 0 & \cdots & 0 \\ p_{2,1} & p_{2,2} & 0 & \cdots & 0 \\ p_{3,1} & p_{3,2} & p_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{b,1} & p_{b,2} & p_{b,3} & \cdots & p_{b,b} \end{pmatrix} = \begin{pmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{R} & \mathbf{T} \end{pmatrix}$$

Apparently, $\mathbf{C} = p_{1,1} = 1 > 0$, $\mathbf{R} = (p_{2,1}, p_{3,1}, \cdots, p_{b,1})' > 0$, and $\mathbf{T} \ne 0$. From Lemma 1, we have

$$\mathbf{P}^{\infty} = \lim_{k\to\infty} \mathbf{P}^k = \lim_{k\to\infty} \begin{pmatrix} \mathbf{C}^k & \mathbf{0} \\ \sum_{i=0}^{k-1} \mathbf{T}^i \mathbf{R} \mathbf{C}^{k-i} & \mathbf{T}^k \end{pmatrix} = \begin{pmatrix} \mathbf{C}^{\infty} & \mathbf{0} \\ \mathbf{R}^{\infty} & \mathbf{0} \end{pmatrix}$$

Here, $\mathbf{C}^{\infty} = 1$ and $\mathbf{R}^{\infty} = (1,1,\cdots,1)'$. We now obtain the stable stochastic matrix $\mathbf{P}^{\infty}$ as

$$\mathbf{P}^{\infty} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}$$

This means that population will eventually fall into the collection with the highest fitness. Thus, (2) in the definition of global convergence is satisfied. ■

4 DYNAMIC PROPERTIES OF THE ALGORITHM

Before we analyze experiment data and compare with other architectures of GA, we should first address the dynamic properties of algorithms with the architecture. We first define the various genetic operators and criteria for evaluating the performance of a given algorithm.

4.1 IMPLEMENTATION AND EVALUATION

Encoding Scheme: Although binary encoding scheme is widely used, we will encode individuals as vectors of real numbers when optimizing continuous functions to avoid the cost of converting binary strings to real numbers, without damaging the convergence properties discussed in the last section. An individual is encoded as an n-dimensional vector $\boldsymbol{v}$=(v_1,v_2,…,v_n), $0 \le v_i \le 1$, i=1,2,…,n. v_i can be transformed when calculating the objective.

Crossover Operator: Given vectors, $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{y} = (y_1, y_2, \cdots, y_n)$, representing two individuals. Their offspring can be obtained by the crossover operator

$$\mathbf{x}' = a \cdot \mathbf{x} + (1-a) \cdot \mathbf{y}$$
$$\mathbf{y}' = a \cdot \mathbf{y} + (1-a) \cdot \mathbf{x}$$

$0 \le a \le 1$ is a random number from a uniform distribution.

Mutation Operator: Given individual, $\mathbf{x} = (x_1, x_2, \cdots, x_n)$. Select x_i , i=1,2,…,n with probability $1/n$ for mutation. If x_k is chosen, we set $x_k' = x_k + 0.1 \cdot \delta$, with $-1 \le \delta \le 1$ is a random number from a uniform distribution. If $x_k' > 1$, let $x_k' = 1$; if $x_k' < 0$, let $x_k' = 0$. Then, x_k' replaces x_k in $\mathbf{x}$.

We now define

$$\alpha = N_1 / N = r_1 \tag{3}$$

$$\beta = N_2 / (N_2 + N_3) = r_2 / (r_2 + r_3) \tag{4}$$

Note that $0 \le \alpha \le 1$ and $0 \le \beta \le 1$. We interpret α as a measure of the relative importance of the exploitation operation and the exploration operation, while β represents the relative importance of the genetic operators and the random operators. Clearly, we can deduce r_1, r_2, r_3 from α and β

$$r_1 = \alpha \tag{5}$$

$$r_2 = (1-\alpha) \cdot \beta \tag{6}$$

$$r_3 = (1-\alpha) \cdot (1-\beta) \tag{7}$$

Evaluating Criteria: Statistical measurements are used to evaluate our algorithms, such as the *average success ratio* R_{AS}, the *average objective calculating number* C_{AVE}, the *first passage time* T_{AFP}, and the *standard deviation of the first passage time* D_{AFP}. We first give some definitions.

Definition 2: An *experiment* is a run of a certain algorithm with the limitation of the maximum iteration number $N_{\max}$. In the n-th iteration of an experiment for some finite positive n, if

$$\begin{cases} |F^* - F_{Best}^n| < \varepsilon \cdot F^* ; F^* \ne 0 \\ |F_{Best}^n| < \varepsilon \quad ; F^* = 0 \end{cases} \tag{8}$$

then, the experiment *succeeds*, otherwise, *fails*. Note that F^* represents the global optimum, F_{Best}^n is the optimum in the n-th generation. For a successful experiment, the first passage time, T_{FP}, is the minimum n that satisfies (8).

Definition 3: Let C be the number of times that the objective is calculated in a successful experiment. The average objective calculating number, C_{AVE}, is the arithmetical average of C over several experiments.

Definition 4: Given M_S successful experiments out of M trials, the average success ratio is $R_{AS} = M_S / M$. Note that R_{AS} is an estimate of P_S, the probability of the algorithm finding the global optimum within a certain time limit.

Definition 5: Given M_S successful experiments out of M trials, the average first passage time, T_{AFP}, is defined as the arithmetical average of the first passage time over successful experiments

$$T_{AFP} = \frac{1}{M_S} \sum_{i=1}^{M_S} T_{FP}^i \tag{9}$$

The standard deviation D_{AFP} of the T_{AFP} is defined as:

$$D_{AFP} = \sqrt{\frac{1}{M_S} \sum_{i=1}^{M_S} (T_{FP}^i - T_{AFP})^2} \tag{10}$$

4.2 SUCCESS RATIO VERSUS (α, β)

To test the performance of algorithms with our architecture and further study their dynamic properties, we use the Rastrigin function (Muhlenbein,1991) defined by

$$R(x) = nA + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i))$$

with $x_i \in [-5.12, 5.12]$, i=1,2,…,n. A is a constant. $R(x)$ is a multi-modal function with the global minimum 0 when x_i=0, i=1,2,…,n and approximately 10n local minima in the range S={ $x_i \in [-5.12, 5.12]$, i=1,2,…,n}. We use A=1.0, n=20, $\varepsilon = 0.01$, and $N_{\max}$=1000 and look for the dependence of R_{AS} on α and β. We change α and β from 0 to 1 in step of 0.05. We run 1000 independent experiments with each parameter combination. Fig.2 shows the relation of R_{AS} versus α and β, where we can define approximately three regions in (α, β) according to R_{AS},

- *The Fail Area*: R_{AS}=0.
- *The Success Area*: R_{AS}=1.
- *The Partial Success Area*: 0< R_{AS} <1.

We may further draw the following conclusions:

1. Our algorithm is safe and robust. A safe algorithm refers to high average success ratio, which means that the global optimum can be found with a fairly high probability. If we find high average success ratio in a wide range of (α, β), then the algorithm is robust as one can choose many combinations of parameters and still run a successful experiment. Fig.2 shows that the algorithm has a large success area and is safe and robust.
2. Several failing areas appear when (i) $\alpha = 0$ with any β; (ii) $\alpha = 1$ with any β; (iii) $\beta = 0$ with any α. Case (i) is similar to population non-overlapped simple GA. Since $\alpha = 0$, no optimum individual ever found is retained in the course of evolution. The algorithm cannot guarantee the global convergence although it has ergodic property (Rudolph,1994). In case (ii), since no genetic operation is performed at all, the algorithm cannot explore any new area in the solution space, the population will remain unchanged and there is no evolution. In (iii), the algorithm depends entirely on the random generated individuals to do exploration. The lack of orienting mechanism results in poor performance, and (iii) is equivalent to random search.

We next introduce the average success area fraction.

Definition 6: Among Q trial values of β that have been tested for a given α, we denote by $Q_{SA}(\alpha)$ the number of trials that yield average success ratio of 1. The average success area fraction under this α is defined as $R_{ASA}(\alpha) = Q_{SA} / Q$, which is shown in Fig.3.

There are four segments in this curve in Fig.3:

- *The Zero segment*: $\alpha = 0$ and $\alpha \geq 0.95$. Here R_{ASA}=0, implying that the global optimum cannot be found without reference to β.
- *The Ascending segment*: $0 < \alpha < 0.05$. R_{ASA} increase rapidly with the increment of α.
- *The Flat segment*: $0.05 \leq \alpha \leq 0.4$. R_{ASA} remains approximately unchanged without reference to α.
- *The Descending segment*: $0.4 < \alpha < 0.95$. R_{ASA} decrease slowly with the decrement of α.

To achieve high performance safely and robustly, we see that α in the range $(0.05 \leq \alpha \leq 0.5)$ will be good.

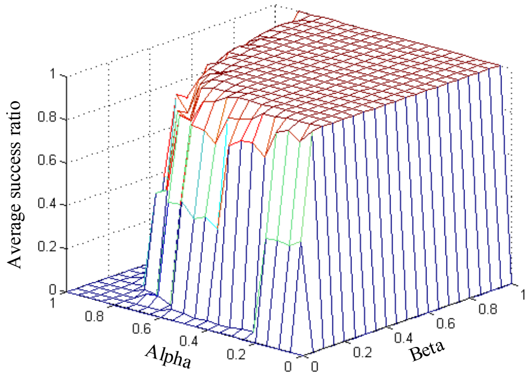


Fig.2: R_{AS} versus (α, β)

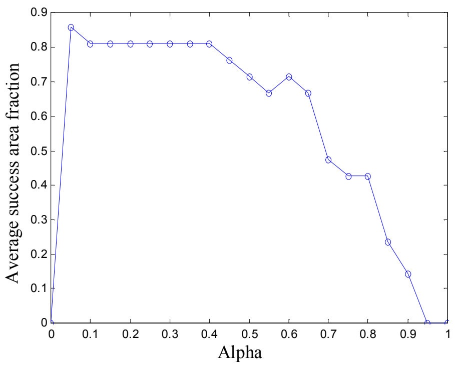


Fig. 3: R_{ASA} versus corresponding α

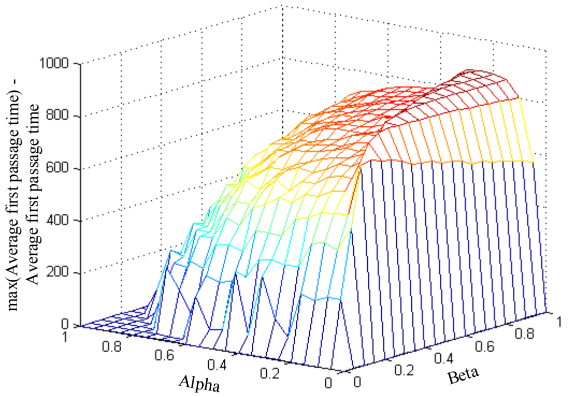


Fig.4: T'_{AFP} versus (α,β)

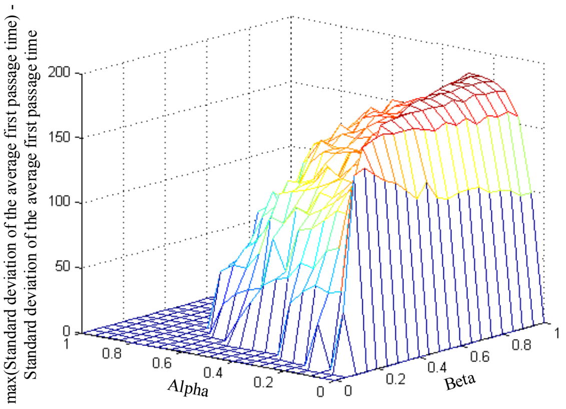


Fig.5: D'_{AFP} versus (α,β)

4.3 FIRST PASSAGE TIME VERSUS (α,β)

The average first passage time $T_{AFP} = T(\alpha,\beta)$ can be obtained under the same condition in 4.2. We introduce $T'_{AFP} = \max\{T_{AFP}\} - T_{AFP}$ for presentation purpose and it is shown in Fig.4. Focusing on the success area, we have two conclusions.

1. α plays an important role in coordinating exploration and exploitation and has great effect on the overall performance. For small α ($\alpha \le 0.05$), the algorithm has a surplus in exploring ability, but deficient in the power of exploitation. For large α ($\alpha \ge 0.3$), the algorithm has a surplus exploiting ability, but deficient in the power of exploration. In either case, unsatisfactory long average first passage times are observed.
2. β tunes the performance slightly. Given α, different β yields different T'_{AFP}, with two regimes observed:
    - *The Unimodal segment*: ($\alpha < 0.3$). T'_{AFP} is a single-peak function of β in this segment. The value of β in the peak increases and the rate of change of the curve after the peak decreases when α increases. We can speculate from these features that random operation enhances exploration to the solution space, thus improving the overall performance. However, when the fraction of the preserved individuals is increased, (i.e., increased exploitation decreased exploration), the contribution of the random operation becomes less important.
    - *The Increasing segment*: ($\alpha \ge 0.3$). T'_{AFP} increases with increasing β and reaches the maximum at $\beta = 1.0$. We can speculate from the features that here a surplus in exploiting power exists, while exploration is insufficient, the algorithm can boost up the exploration only by increasing the effect of genetic operators.

4.4 STANDARD DEVIATION OF THE FIRST PASSAGE TIME VERSUS (α,β)

We can measure the standard deviation of the first passage time $D_{AFP} = D(\alpha,\beta)$, and define as before $D'_{AFP} = \max\{D_{AFP}\} - D_{AFP}$ shown in Fig.5.

Again let's focus on the successful area. First of all, Fig.5 is very similar to Fig.4 for T'_{AFP} versus (α,β). For small α ($\alpha \le 0.05$), the algorithm has a surplus in exploring power but deficient exploiting ability, while the contrary result is associated with a large α ($\alpha \ge 0.3$). Thus, the algorithm presents unsatisfactory large deviations in both cases. Since the existence of the deviation implies an inherent risk associated with the use of the algorithm, we like to have small D_{AFP} for the algorithm with the same R_{AS} and T_{AFP}. Similar to T'_{AFP} , D'_{AFP} also possesses a uni-modal area and an increasing area.

4.5 CONCLUSIONS ON THE DYNAMIC PROPERTIES

In sub-section 4.2 to 4.4, we have analyzed the properties of the average success ratio, the average first passage time and its standard deviation versus α and β. We see that an optimum situation for a genetic algorithm with our architecture occurs when the balance of the exploration and the exploitation is achieved. In this situation, the algorithm runs in the success area of the average success ratio, and at the same time is located in the uni-modal area, or in the increasing area that is close to the uni-modal area of the average first passage time and its standard deviation. We can also speculate that our genetic algorithms will perform best when the exploration and the exploitation are coordinated, so that the majority of genetic operations should be maintained and properly used, while attentions to the random operations can also contribute. Based on these analyses, we expect that the area with $0.05 \le \alpha \le 0.5$ and $0.5 \le \beta \le 1.0$ is the region in the parameter space that yields better performance as the exploration and the exploitation can be well balanced.

5 EXPERIMENTS AND RESULTS

We compare the performance of some typical genetic algorithms with our new algorithms (For brief, acronym NGA is used for GAs with our architecture). These algorithms include the canonical genetic algorithm (CGA), the elitist genetic algorithm (EGA) and the steady genetic algorithm(SGA) (Holland,1975), (Goldberg,1989) and

(Pan,1998). Testing functions include Rastrigin function (Muhlenbein,1991), Griewangk function (Griewangk,1981), and Schaffer function (Schaffer,1989).

Canonical Genetic Algorithms: Simple GA with non-overlapped population over generations. In each generation, certain reproduction scheme is adopted and N individuals are selected from $P(t)$ and crossover and mutation are then performed on them to generate the population $P(t+1)$. Finally, $P(t+1)$ replaces $P(t)$ entirely.

Elitist Genetic Algorithms: A variable E is used to store the best individual ever found thus far. After the population $P(t+1)$ is generated by performing the same operations as those in the canonical genetic algorithm, compare E with B, the best individual in $P(t+1)$. If E is better, choose one individual randomly from $P(t+1)$ and replace it with E, otherwise update E with B.

Steady Genetic Algorithms: After a new population $Q(t)$ is generated by performing the same operations as those in the canonical genetic algorithm, select N individuals from the population that consists of $P(t)$ and $Q(t)$ to form the population $P(t+1)$. Here, N best individuals is selected out of $2N$ individuals in $P(t)$ and $Q(t)$ to form $P(t+1)$.

Rastrigin Function: As is described in 4.2.

Griewangk Function: It is defined as,

$$G(x) = \frac{1}{4000}\cdot\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Where, $-600 \le x_i \le 600$. The global minimum of this function is $x_i=0$, $i=1,2,\ldots,n$, while its local minima are $x_i \approx \pm k\cdot\pi\cdot\sqrt{i}$, $i=1,2,\ldots,n$.

Schaffer Function: It is defined as,

$$S(x) = 0.5 - \frac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{[1 + 0.001\cdot(x_1^2 + x_2^2)]^2}$$

with $-100 \le x_{1,2} \le 100$. Its global maximum is (0,0), while there exist uncountable local maximum within a distance π apart from the global maximum. The feature of rapidly varying multi-modal with global optimum surrounded by local ones makes it difficult to locate the global maximum.

For all algorithms, we set N=100, P_c=1.0, P_m=0.8, and $N_{\max}$=10000. For NGA, α=0.25 and β=0.95 and run each algorithm 10000 times over each function to get statistics such as the average success ration R_{AS}, the average objective calculating number C_{AVE}, the average first passage time T_{AFP} and its standard deviation D_{AFP}. All these statistics are listed in Tab.1~3. These statistics manifest a large variety of behavior due to the diversity of the solution spaces of these functions. Nevertheless, we may draw the following conclusions.

1. The canonical genetic algorithm substitutes the whole population with newly generated individuals in each generation, causing probably the lost of some existing excellent genes. Since the operation of crossover and mutation may damage some excellent individuals and generate some bad ones, it is hard for the algorithm to find the global optimum.
2. The elitist genetic algorithm has the property of global convergence (Rudolph,1994). However, it suffers from a deficiency of the exploring ability. The algorithm needs a very long period to converge to the global optimum, causing an unacceptably large average first passage time.
3. The steady genetic algorithm adopts a simple trade-off between exploration and exploitation, thus achieving generally good results. Nevertheless, since the trade-off pays no attention to the coordination between the exploration and the exploitation to the solution space, the average success ratio is low while the average first passage time is long when the algorithm optimizes some complex functions such as the Griewangk and the Schaffer function. This statistics suggests that the steady genetic algorithm is likely to be trapped into local optima.
4. The algorithm with our architecture (NGA) exhibits excellent performances when optimizing these functions. The introduction of controlling parameters allows a better balance between exploration to the problem's solution space and the exploitation of better solutions, thus getting excellent results.
5. In CGA, EGA and SGA, N new individuals are generated in each generation, thus objective value should be calculated N times. In contrast, in NGA, only $(1-\alpha)\cdot N$ new individuals are generated and evaluated in each generation. This yields considerable saving in computational cost and our NGA algorithms can perform more iteration with the same processor time.

Tab.1: Rastrigin function with ε=0.0001

Algorithm	R_{AS}	T_{AFP}	D_{AFP}	C_{AVE}
CGA	0.00	—	—	—
EGA	1.00	2205.80	325.35	220580
SGA	1.00	753.20	193.59	75320
NGA	1.00	296.26	37.87	22220

Tab.2: Griewangk function with ε=0.01

Algorithm	R_{AS}	T_{AFP}	D_{AFP}	C_{AVE}
CGA	0.00	—	—	—
EGA	0.00	—	—	—
SGA	0.63	1700.58	680.72	170058
NGA	1.00	758.20	189.35	56865

Tab.3: Schaffer function with ε=0.0001

Algorithm	R_{AS}	T_{AFP}	D_{AFP}	C_{AVE}
CGA	0.00	—	—	—
EGA	0.89	896.88	474.56	89688
SGA	0.33	381.05	268.42	38105
NGA	1.00	365.63	122.88	27423

6 CONCLUSIONS

We have introduced and analyzed a new genetic algorithm (NGA) and demonstrated with sufficient statistics over several well-known test functions that our algorithm is superior to existing ones. We also provide a theoretical understanding of the good performance underlying our algorithms. In the NGA architecture, a preserved part, a reproduced part and a randomized part constitute the new population in each generation. The amount of individuals in each part is controlled by two parameters r_1 and r_2. In applying the algorithm, the preserved part contains copies of excellent individuals in the parent generation. Performing crossover and mutation on individuals selected from the parent generation creates the reproduced part. The randomized part consists of individuals randomly generated. From the viewpoint of exploring the solution space and exploiting the information in past search, our attention to exploitation is measured by the amount of the preserved individuals. The attention we pay to the effect of various genetic operations while exploring the solution space is measured by the amount of the reproduced individuals. Finally, the attention we pay to the effect of getting trapped in local optima is represented by the amount of the randomly generated individuals. Since we fix the scale of the population, only two numbers (α and β) or (r_1 and r_2) are needed to parameterize these three portions that make up the entire population.

Experiments have been designed to analyze the dynamic properties of algorithms with our architecture and to test the performance of algorithms with the architecture while doing optimization over complex multi-modal functions. Our analyses of statistics are quite extensive, including the average success ratio, the average first passage time and its standard deviation. Since parameters are introduced to coordinate exploration and exploitation in the solution space, our new algorithm is more flexible and robust than CGA, EGA and SGA. To summarize, our algorithm has the good features of economy, robustness, and efficiency.

Economy: There are two aspects. (1) All the individuals in the parent population are sorted according to their fitness in descending order, before generating the preserved sub-population. The result of the sorting can be used to generate the parent individuals while doing crossover and mutation. (2) All the newly generated individuals, including the reproduced ones and the randomly generated ones, will be accepted, implying that we do not waste any effort in their calculation of the objective.

Robustness: There exists a large success area in our new genetic algorithm. This indicates that our algorithm can find the global optimum with a maximum probability, or in another word, with the minimum risk. This is a strong indication that our new genetic algorithm is robust.

Efficiency: From the point of view of T_{AFP}, the average first passage time, our new genetic algorithm outperforms some typical genetic algorithms in various experiments. This result is explained in the paper and we understand this efficiency as a manifestation of the good coordination and balance between exploration, exploitation, and between directed search and the careful avoidance of local traps. From the point of view of D_{AFP}, the standard deviation of the average first passage time, our new genetic algorithm maintains a small deviation while getting satisfactory T_{AFP}. This indicates little divergence while optimizing.

Finally, adaptive features will be introduced into the new genetic algorithm described in this paper to achieve more flexibility in dealing with a much wider class of problems in future publication (Jiang et al, 2001).

References

J.H. Holland (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

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

Z.J. Pan (1998), L.S. Kang, Y.P. Chen, *Evolutionary Computation*. Beijing: Tsinghua University Press, Guangxi Science and Technology Press. (In Chinese).

K.Y. Szeto, K.H. Cheung (1998). Multiple Time Series Prediction Using Genetic Algorithms Optimizer. *Proc. of the Int'l. Symposium on Intelligent Data Engineering and Learning*, Hong Kong, IDEAL'98, 127-133.

K.Y. Szeto, P.X. Luo (1999). Self Organized Genetic Algorithm in Forecasting Stock Market. *Proc. of the Sixth International Conference 'Forecasting Financial Markets'*, London, 26-28 (Invited talk at the FFM'99).

D.C. Hu, R. Jiang, Y.P. Luo (1999). An Adaptive Classifier System Tree for Extending Genetics-Based Machine Learning in Dynamic Environment. *Journal of Artificial Life and Robotics* (Accepted as an invited paper).

C.A. Ankerbrandt, B.P. Buckles, F.E. Petty (1990). Sense Recognition Using Genetic Algorithms with Semantic Nets. *Pattern Recognition Letters*, **11**: 285~293.

M. Srinivas, L.M. Patnaik (1994). Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE, Trans. On System, Man and Cybernetics* **24**(4): 656~667.

G. Rudolph (1994). Convergence Analysis of Canonical Genetic Algorithms. *IEEE Trans. Neural Networks, special issue on Evolutional Computing*, **5**(1): 96~101.

B. Dinabandhu, C.A. Murthy, K.P. Sankar (1996). Genetic Algorithm with Elitist Model and Its Convergence. *International Journal of Pattern Recognition and Artificial Intelligence*, **10**(6): 731~747.

J. Suzuki (1995). A Markov Chain Analysis on Simple Genetic Algorithm. *IEEE Trans. System, Man and Cybernetics*, **25**(4), 655~659.

M. Iosifescu (1980). *Finite Markov Processes and Their Applications*. Wiley, Chichester.

H. Muhlenbein, M. Schomish, J. Born (1991). The parallel Genetic Algorithm as Function Optimizer. *Parallel Computing*, **17**: 619~632.

A.O. Griewangk (1981). Generalized Descent for Global Optimization. *JOTA*, **34**: 11~39.

J.D. Schaffer, R.A. Garuana, L.J. Eshelman, et al (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. *Proc. of the 3rd Int'l. Conf. on Genetic Algorithms*, Los Altos.

R. Jiang, K.Y. Szeto, Y.P. Luo, D.C. Hu (2001). Adaptive Genetic Algorithm: An Entropy Estimating Approach (preprint).

An Adaptive Genetic Algorithm

Eric Kee, Sarah Airey and Walling Cyre
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA 24061

Abstract

This paper presents an adaptive genetic algorithm that learns to adjust some of its parameters for rapid solution based on the current state of the population. The mapping from population states to parameter values is learned during a training phase that is followed by an execution phase during which the algorithm uses the learned mapping to solve the problem. The algorithm learns which crossover rate, mutation rate, and fitness scale factor produce the best fitness growth for the problem. Two variations of the method were tested on the DeJong problem suite, and in some cases produced a 66% improvement over a non-adaptive approach.

1 INTRODUCTION

The parameters that control the development of a solution population in a genetic algorithm (GA) are the driving force behind the algorithm's success or failure. Although studies have discovered parameter values that produce rapid results on average, we hypothesize that optimum parameter settings vary depending upon a problem's fitness landscape. For example, a smooth, continuous, fitness landscape with one maximum should be easily solved without mutation, while a rough landscape with many local maxima may require high population mutation to explore the many maxima before the solution is found. If optimum parameters are dependent upon the shape of a fitness landscape, we conjectured that an **Adaptive Genetic Algorithm** (**AGA**) could investigate its solution space to derive optimum parameter settings while the population moves through the fitness landscape. Our work attempts to devise a method for real-time adaptation of GA control parameters—crossover probability, mutation probability, and power scaling factor—in order to maintain consistent positive fitness growth.

If an AGA is to modify its parameters based upon the fitness landscape, it must have a method for estimating the characteristics of that landscape and how the population is distributed over that landscape. This work uses three measures on the population to estimate the state of the population in a fitness landscape: 1) the rate of change of the maximum fitness (ΔF_m), 2) the variance of the population's fitness values (σ^2_f), and 3) the diversity of its chromosomes (σ^2_p). ΔF_m was selected as a measure of how close the population is to local maxima, and σ_f and σ_p provide measures of the population's diversity within the landscape. The triple of values, (ΔF_m, σ_f, σ_p), for a given population will be called the **state vector** for that population. The objective, then is to develop a mapping from state vectors to parameter values for the genetic algorithm. Two difficulties arise when defining a set of population state vectors and corresponding genetic parameter settings. First, the number of values state vectors may have can be very large. Second, the AGA must know the optimum genetic parameters for each state vector.

To reduce the number of state vector values, a set of sub-ranges are defined for each real-valued component of a state vector. In this paper, three ranges: high, medium and low, are defined for each state vector component, and each range is assigned a representative value, called a parameter **level**. If the ranges and distributions of the state vector components are known *a priori*, the levels can be readily specified. Otherwise, random individuals can be generated and evaluated to determine sample values for each component of the state vector. The second problem is to determine the correlation between state vector levels and optimal parameter values. This was accomplished in the present work by learning during execution of the genetic algorithm.

2 RELEVANT WORK

Many attempts have been made to identify optimum values for the control parameters used in genetic algorithms. Initial work by DeJong suggested ideal values for population size and probabilities for crossover and mutation based on how the values performed on the now standard DeJong test suite (DeJong, 1975). These suggestions were more finely tuned by Grefenstette (Grefenstette, 1986) who employed a “meta-level genetic algorithm” to optimize six control parameters of another GA, including population size, crossover and mutation rates, selection and replacement strategies, and fitness scaling (Grefenstette, 1986). However, both of these findings yielded static parameter values for the entire run, but the optimal setting for the control parameters is likely to change as the run progresses. In contrast, our AGA continually adjusts the parameters during runtime to produce the most positive growth.

Several strategies have been developed to provide adaptive parameters that change during an algorithm’s execution. There are two general approaches: co-evolutionary methods that encode the parameters into the chromosomes and allow them to evolve with the solutions and learning rule methods that continually evaluate the performance of the operators and adjust their values accordingly (Tuson, 1995). In co-evolutionary methods, individuals with poor operator settings are weeded out of the population. Bäck showed some success by encoding the mutation operator into the chromosome. The mutation rate adapted to the theoretical optimum for the problem considered (Bäck, 1992). Similar methods for adjusting crossover rate and point have shown limited performance improvements (Tuson, 1995). The AGA method presented here reduces overall run time required to find a solution by up to 66%.

One well-known learning rule method is the *adaptive operator fitness* developed by Davis (Davis, 1989). The adaptive algorithm records the parent and operator for each offspring chromosome. When an offspring is better than the current best chromosome, the operator used in its creation is credited in proportion to the improvement in fitness. Discounted proportions of this reward are allotted to the operators that produced the offspring’s parents, grandparents, and so on. After an interval of generations, the probability of an operator is recalculated based on its previous fitness and credits received over the most recent interval. This method proved effective at enhancing performance of a GA on certain problems (Davis, 1989). Another variation of this adaptation method designed by Julstrom updates the probability of an operator after every generation (Julstrom, 1995). Operator probabilities are computed entirely from the operator’s most recent contributions to the algorithm’s performance as measured by how many offspring it produced that are better than the population median. This method also showed some improvement in solving certain problems, including the traveling salesman problem (Julstrom, 1995). However, some suggest that the rate of adaptation of the parameters might not match the rate of adaptation of the population—that it takes too much time for the most effective operators at a certain point to build up the fitness that increases their probability to the optimum level (Mitchell, 1999). Our work attempts to eliminate the delay by tying parameter values to the current state of the population.

3 DESIGN APPROACH

We hypothesized that the performance of a genetic algorithm during a given generation is dependent on the distribution of the population in the fitness landscape and the values of the algorithm’s control parameters in that generation. Three characteristics—the level of diversity among the population individuals, the diversity of their fitness’, and the rate of change of the fitness—seem to reveal important information about the state of population in the fitness landscape. Based on this, the notion of a state vector was defined as follows:

The three components of a population **state vector**, (ΔF_m, σ_f, σ_p), are

> ΔF_m, the *fitness velocity is* the rate of change of the best fitness over a specified number of consecutive generations,
>
> σ^2_f, th*e fitness variance* is the statistical variance of the fitness values of the entire population, and
>
> σ^2_p, the *population variance* is a measure of population diversity determined by the hamming distances among the chromosomes of the top N individuals.

Of the many control parameters used in a genetic algorithm, three parameters were selected here: crossover probability (Pc), mutation probability (Pm), and the power fitness scaling factor (α) used to control selection pressure. The values of these three parameters, (Pc, Pm, α) are called a **control vector**.

Two mechanisms were used to represent the mapping between population state vectors and control parameter values. One approach used a table, and the other used a set of rules. In each case, the ranges of the state vector components were partitioned into high, medium and low subranges in order to reduce the size of the state vector table to $3^3 = 27$ entries, and the number of rules, to 27. The subranges ranges of state vectors were estimated by running a sample population for a limited number of generations and recording the population’s state vectors after each generation. This data was sorted and partitioned into three subranges.

In the table-based approach, the mapping between state vectors and GA control parameter values is maintained in a state vector table, where each entry is a control vector. In the rule-based approach, the control parameter values are quantified to high, medium and low values, resulting in only 27 possible control vectors. For each of the 27 state vectors, a rule specifies the best control vector. An example rule is “if the state vector is (medium, low, high) then set the control vector to (medium, low, medium).

In both approaches, the adaptive algorithm has a training phase, followed by an execution phase. During the training phrase, the state table or rules are developed, and during the execution phase, the algorithm uses the table or rules to search for a solution. A training phase consists of a number of training epochs. A **training epoch** consists of a number of generations of the genetic algorithm. The execution phase simply continues the search without further adaptation of the table or rules.

3.1 TABLE-BASED TRAINING

State vector table training uses a table having an entry for each of the twenty-seven possible state vector combinations with each entry representing a control vector. Each control vector was initialized to (.95, .01, 1). The initial crossover and mutation probabilities are the optimum values determined by Grefenstette's meta-level GA experiments (Pc=0.95 and Pm=0.01) (Mitchell, 1999). The power scaling factor, α, was initialized to 1 for no scaling. During a table training epoch, the algorithm maintains two copies of the current population—a *constant population* (C) and a *test population* (T). Initially, the constant population is equivalent to the test population, and the state vector is calculated for the common population. Population C uses the control vector listed in the current state vector table, while population T uses parameter values that have been randomly varied from those given in the state vector table. Then, each population is subjected to N generations of evolution. Once the two populations have evolved for N generations, the population with the higher average fitness is saved as population C and its control vector is entered into the state vector table. This process of evolving populations C and T, determining which has higher fitness, and updating the state vector table with new control vectors, is repeated for a selected number of training epochs. Through this procedure of modifying control vectors and evaluating their effectiveness, the GA can fine-tune the control vectors to values that will yield the most desirable fitness growth.

3.2 RULE TRAINING

Rule training uses a n expanded state vector table to determine the control vectors for the various population state vectors. The expanded table has a set of 27 entries for each state vector, with each entry corresponding to a possible rule consequent, such as a (medium, low, medium) control vector. The value of each entry is the probability that that consequent is optimal for that state vector. Initially, all entries for a state vector are $1/3^3$.

During training, a control vector (rule consequent) is selected probabilistically from the expanded table for the given state vector of the population. A training epoch is then run with this control vector and the probabilities of the entries for the state vector are adjusted based on the rate of change in fitness. If the rate of change of the fitness increased compared to that of the immediately previous generation, then the entry for the last combination applied is increased for that state vector. Since some changes, such as an increase in mutation, might not produce immediate gains in fitness, but instead offer improvement over the long-term, the entry of each of the last few combinations applied are also increased for their respective state vectors. Entries of the other combinations for those state vectors are decreased equally to punish unsuccessful combinations. The sum of entries for a given state vector are constrained to be unity, so the entries may be thought of as probabilities of selecting a parameter combination given a population state. Also, no entry in the table was permitted to fall below a specified minimum so each control vector has some probability of being selected in any generation. At the end of the training period, the rules used during the execution phase are formed by taking the control vector having the highest entry for the given state vector as the consequent of the rule.

4 EXPERIMENTS AND RESULTS

A number of preliminary experiments were run on convenient problems to get a sense of useful ranges for the number of generations per epoch and the number of training epochs that should be used in the training phase. It appeared that large numbers of epochs were beneficial, but that epochs should have few generations. Experiments were also run to determine suitable subranges for high, medium and low classification of the state vectors.

The evaluation experiments were performed on a 450 MHz Pentium III machine with the adaptive and non-adaptive genetic algorithms on DeJong's test suite (DeJong, 1975). These functions are shown in Table 1. Three of the functions implemented represent one aspect of a difficult fitness landscape: the solution to DeJong's second function lies on a very sharp ridgeline, the solution to DeJong's third function lies at the bottom of a three dimensional stair-like surface, and the solution to DeJong's fourth function lies at the top of a paraboloid whose surface is distorted by Gaussian noise. DeJong's first function is a paraboloid and is intended to be simple.

Table 1. DeJong Functions

Function	*Mathematical Expression*
1	$f_1(x) = {}_{i=1to3}\Sigma x_i^2$
2	$f_2(x) = 100(x_1^2-x_2)^2+(1-x_1)^2$
3	$f_3(x) = {}_{i=1to5}\Sigma[x_i]$
4	$f_4(x) = {}_{i=1to30}\Sigma i x_i^4 + N(0,1)$

4.1 TABLE-BASED AGA EXPERIMENTS

Using the simple AGA, we ran a series of tests on each function. For each DeJong function, 51 random initial populations were run with both the AGA and the non-adaptive GA. In each pair of runs, both GA's started with the same initial population, as well as the same initial control parameter values: Pc=0.95 and Pm=0.01, and α=1. Each run used 100 training epochs of one generation each. Table 2 shows how the AGA improved over the non-adaptive GA, both in number of generations and run time required to find a solution. The second column of the table shows the average percentage reduction in the number of generations required to find a solution with the adaptive algorithm. For each case, the total number of generations was reduced by at least 50%. The standard deviation of the reductions are given in the third column. The average time reductions with the adaptive algorithm are shown in column 4. Runtime reductions are also shown because the AGA training time is not reflected in the number of generations. The average run times were reduced on most functions, but actually increased in one case. In addition to improved average run times and solution generations, the AGA produces more reliable and consistent results across all test seeds. In all cases, the standard deviation of both run time and generations-to-solution is much smaller than in the non-adaptive GA. The final column shows the percentage of the run time that was spent in the training generations. For most function this was relatively small, but became significant in Function 3.

In Figures 1-4 below, the runtimes in seconds required for the non-adaptive GA to reach a solution are plotted against the those for the AGA. Each point represents one of the 51 the initial populations used. For Function 1 and 2 the adaptive algorithm is clearly faster for nearly all the initial populations tested. Figure 3, however, shows that the adaptive algorithm is as liable to be slower as it is to be faster on Function 3, which has the simplest fitness landscape. On the fourth function, the adaptive algorithm is marginally faster in most cases, but is clearly faster in others. From these experiments, the AGA performs best when faced with a complex problem—this can be attributed to the overhead inherent to the simple training method.

Table 2. Percentage Reduction in Generation and Run Time Yielded by AGA over GA

Function	*Avg. Generations*	*σ. of Generations*	*Avg. Run Time*	*σ of Run Time*	*Training Time*
DeJong 1	76.0	67.7	56.1	45.1	11.4
DeJong 2	81.3	81.7	66.1	69.7	9.6
DeJong 3	63.3	54.6	-3.4	25.8	44.5
DeJong 4	54.8	59.0	19.9	32.4	7.5

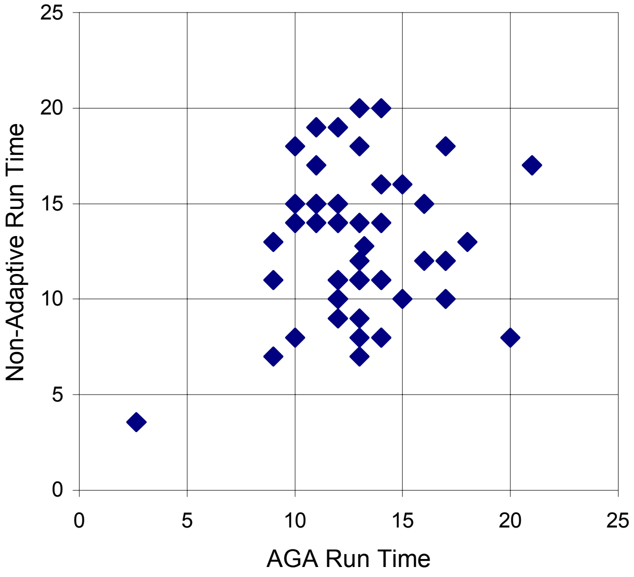


Figure 1. Table-Based Runtimes on Function 1.

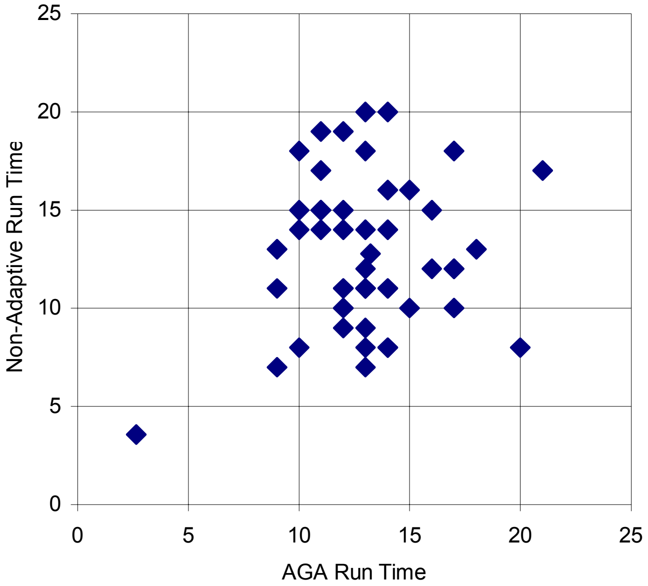


Figure 2. Table-Based Runtimes on Function 2.

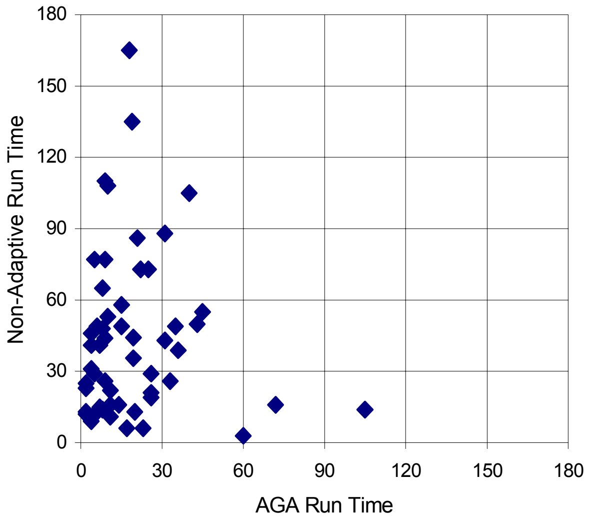


Figure 3. Table-Based Runtimes on Function 3.

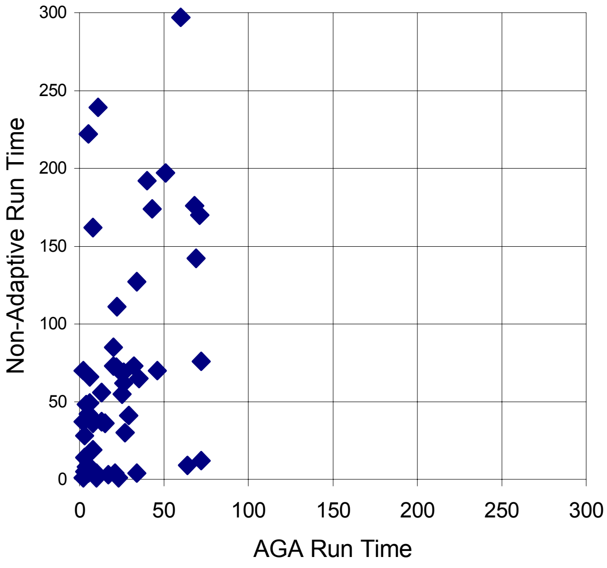


Figure 4. Table-Based Runtimes on Function 4.

4.2 RULE-BASED AGA EXPERIMENTS

The quantified control parameter values used to form the 27 control vectors were defined based on the earlier experimental results. They are given in Table 3 below.

Table 3. Control Parameter Values

Parameter Value	*Pc*	*Pm*	*α*
High	0.95	0.95	4
Middle	0.50	0.50	1
Low	0.05	0.05	0.25

In the rule-based approach, a training epoch consisted of one generation. To begin training, each entry in the state table held was initialized to 0.037 (= $1/3^3$). For each training epoch, the state vector was determined from the population and the control vector was selected probabilistically for the next generation. If the rate of change of the best fitness had increased compared to that of the immediately previous generation, then the entries for each of the last ten parameter choices were increased by 0.05. Entries for the other parameter values for those states were decreased equally to provide this reward. However, no probability was decreased below 0.005. So, no control vector ever evolved a probability beyond 0.87. At the end of training, the rules were compiled using the control vector with the highest probability for each state vector. Once the training was complete, the algorithm continued executing using the learned rules. The training generations were included in the count of the total number of generations required to reach a solution.

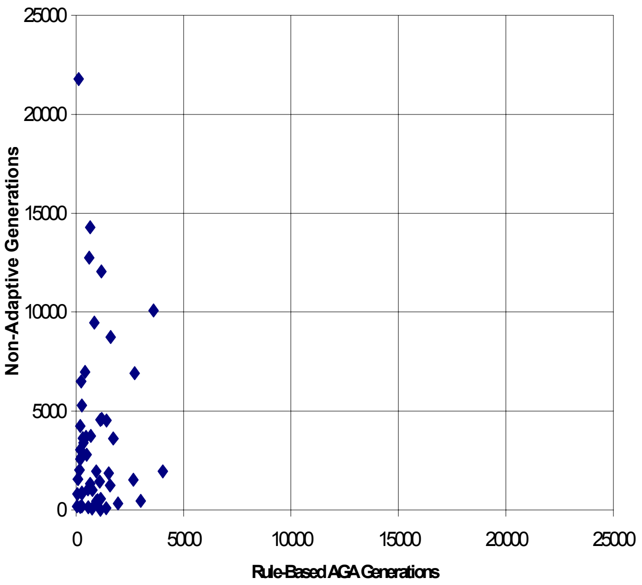


Figure 5. Rule-Based Generations on Function 2.

The rule-based AGA was tested extensively on only two of the DeJong problems, Functions 1 and 2. Results from the rule-based AGA were compared with those from the standard non-adaptive GA, where both algorithms started with identical initial populations. For each function the AGA was run with 250, 500, 750, and 1000 training epochs/generations, but 500 generations appeared to yield the best training, so those results are presented here. Since the training generations required no additional computation time compared to the normal generations, the running time is directly proportional to the number of generations required to reach a solution.

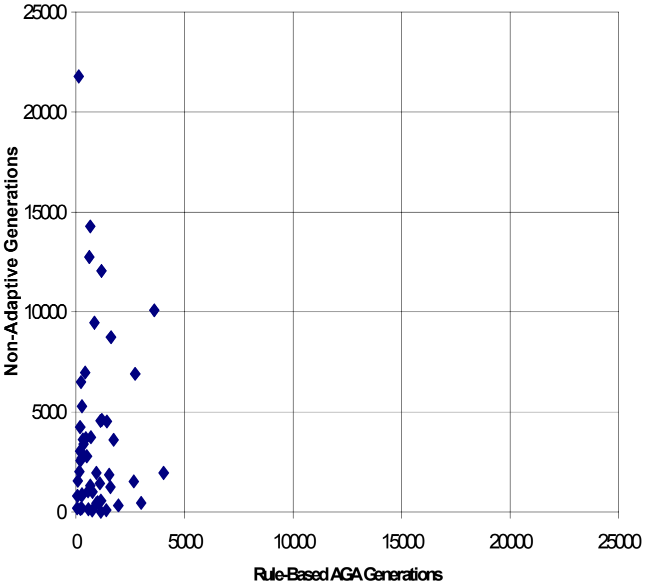


Figure 6. Rule-Based Generations on Function 1.

On Function 2, the saddle, the AGA reduced the number of generations required to solve the problem by 74% over the non-adaptive GA, that is, an average of 961 generations compared to 3693 generations. The run time reduction on this problem is slightly better than the reduction experienced with the table-based AGA. The results of these runs are shown in Figure 5. However, on Function 1, the AGA performed 85% more generations than the non-adaptive GA, as shown in Figure 6. On this problem, the AGA required an average of 9236 generations to reach a solution, while the non-adaptive GA solved it in an average of only 1226 generations. The rule-based approach even failed to solve the problem from initial populations that presented no challenge to the non-adaptive GA. The rule-based AGA also performed relatively poorly on Functions 3 and 4.

The rule-based AGA did evolve rules for each of the 27 states. Some typical rules that were learned are shown in Table 4. Some of the rules seem to be valid, such as increasing the mutation rate when population variance, fitness variance, and fitness rate of change are all low. However, the rules evolved only proved effective in solving one of the four functions. It fared well on the most difficult of the four problems, but much worse on the others.

The relatively poorer performance of the rule-based algorithm is not altogether surprising. Both approaches develop a mapping from 27 state vector ranges to 27 control vectors. The table-based approach allows the learned control vectors to be fine tuned, whereas the rule-based approach leads to coarsely quantified values. In addition, the rule-based approach did less exploration than the table-based approach during training, since the table-based approach evolved two populations during that time.

Table 4. Example Control Parameter Rules Learned

State Vector			*Control Vector*		
ΔF_m	σ^2_f	σ_p	Pc	Pm	α
Low	Low	Low	Low	High	Low
Med.	High	Low	Med.	High	Low
Low	High	Med.	Med.	Med.	Med.
Low	Low	High	High	Low	High
Med.	Low	High	Med.	Low	Med.
Low	High	High	High	Low	Low
Low	Low	Low	Low	High	Low
Low	Low	High	High	Low	High

5 CONCLUSIONS

The concept of a *state*—and its relationship to the fitness landscape—as applied to the population of an adaptive genetic algorithm, *does* appear related to the appropriate genetic control parameters needed to find a solution optimally. Although our more successful training method was not complex, it produced significant performance increases over its non-adaptive cousin. Further research need to be done in a method for deciding when an adaptive versus a non-adaptive approach should be applied. That is, the algorithm need to be able to adjust its training plan. Also, the poor results on some functions with the rule-based method needs to be diagnosing more accurately.

Acknowledgments

The research reported here was supported in part by the National Science Foundation, Grant MIP-9707317.

References

Bäck, Thomas, "Self-adaptation in genetic algorithms," *Proc. First European Conf. Artificial Life,* Paris, 263-271, December 11-13, 1991.

Davis, L. D., "Adapting operator probabilities in genetic algorithms," *Proc. of the Third International Conference on Genetic Algorithms,* San Mateo, CA, 61-69, June 4-7, 1989.

De Jong, K. A., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems,* Ph. D. Thesis, University of Michigan, Ann Arbor, 1975.

Grefenstette, John J. 1986. "Optimization of Control Parameters for Genetic Algorithms," in Bill P. Buckles and Frederick E. Petry, eds. *Genetic Algorithms,* IEEE Press, 1992.

Julstrom, Bryant, "What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm," *Proc. of the Sixth International Conference* *on Genetic Algorithms,* Pittsburgh, PA, 81-87, July 15-19, 1995.

Mitchell, Melanie, *An Introduction to Genetic Algorithms,* MIT Press, Cambridge, MA, 1999.

Tuson, Andrew Lawrence, *Adapting Operator Probabilities in Genetic Algorithms,* M.Sc. thesis, University of Edinburgh, UK, 1995.

A Double-Stranded Encoding Scheme with Inversion Operator for Genetic Algorithms

Paul J. Kennedy
Department of Software Engineering
Faculty of Information Technology
University of Technology, Sydney
PO Box 123 Broadway NSW 2007 AUSTRALIA
paulk@it.uts.edu.au

Thomas R. Osborn
The NTF Group
Decision Support Consultants
Level 7, 1 York Street
Sydney NSW 2000 AUSTRALIA

Abstract

A new double–stranded encoding scheme based on the DNA molecule is described. Double–stranded encodings allow the representation of more schemata than single–stranded encodings containing the same number of bits. This allows GAs to search more efficiently. Experiments showed faster initial search by double–stranded encodings than with single–stranded genomes. Later, however, inter–strand constraints emerge that reduce the efficiency of search. Techniques to alleviate the constraints are discussed. Use of a double–stranded encoding allows simple implementation of the inversion genetic operator. Results show that higher rates of inversion lead to earlier discovery of high fitness and increased convergence of the population towards it.

1 Introduction

A theme of much work in evolutionary computation is the addition of increased levels of biological realism into models. Examples start back at the very dawn of the field with Rosenberg's evolution of models of single–celled organisms (Rosenberg, 1967) and Weinberg's computer simulation of a living cell (Weinberg, 1970) and continue through to recent times with the modelling of biological mechanisms such as gene expression (Kennedy and Osborn, 2000b) (O'Neill and Ryan, 2000) (Kargupta, 2000) and transposons (Simões and Costa, 2000).

In a similar vein, we propose a new encoding scheme based on the deoxyribonucleic acid (DNA) molecule. DNA is the primary storage repository of genetic information in biological cells. Most of the time, each DNA molecule takes the form of two complementary strands (polynucleotides) wound into a double helix. In effect, this means that DNA molecules store the same pieces of information twice. The benefit that DNA gains from being double–stranded is a reduction of errors in the stored genetic information.

Reducing the number of errors caused by thermal fluctuations is a task that is necessary as the result of the physics of the real world. It does not, in general, apply to computer simulations or evolutionary computation. We usually need to make an effort to introduce errors and randomness. This would suggest that modelling double strands in simulated DNA is not useful.

We built a biologically inspired model of a single–celled organism (Kennedy, 1998) (Kennedy and Osborn, 2000a). Cell simulations were evolved so as to find cells able to live in a simulated environment. As a part of this work we developed a double–stranded encoding. Our motivations were exploratory and we aimed to be faithful to biology.

It transpired, however, that this double–stranded encoding provided two benefits other than increased stability that, of course, is not exhibited in our model (as we do not model random thermal fluctuations to any strong degree): (i) we found that double–stranded encodings allow a broader and faster search of the problem space than single–stranded encodings containing the same number of bits; and (ii) a simple implementation of the inversion genetic operator is possible with double–stranded encodings and use of this operator increases convergence of the population to higher fitness. However, using a double–stranded encoding eventually causes additional constraints to emerge between genes on the strands. The effect of these constraints is to slow the search. We discuss methods to alleviate the problems caused by the inter–strand constraints.

The rest of the paper is organised as follows. In section 2 the molecular biology of molecules involved in the storage of genetic information is introduced. Next the double–stranded encoding and its operation with genetic operators (including inversion) is described. Following this we examine the double–strand encoding with respect to schemata represented. Finally we present results for experiments comparing single and double–stranded encodings and experiments that vary the rate of inversion.

2 Biological Motivations

Biological cells use two kinds of macromolecule to store genetic information: DNA (deoxyribonucleic acid) and RNA (ribonucleic acid) (Alberts et al., 1994). Each of these polynucleotides have different uses in cells which is reflected by characteristics of their molecular shape. Double–stranded DNA is the permanent store of genetic information in the cell. DNA molecules guide the building of particular RNA molecules by acting as a template. Single–stranded RNA molecules on the other hand are used primarily to build protein molecules, both as a template to guide the creation of proteins (messenger RNAs) and as RNA catalysts (for example, ribosomes). Both RNA and DNA are long unbranched polymer chains of nucleotides.

Nucleotide molecules consist of a 5–carbon sugar bound to a phosphate molecule (from the $5'$ carbon of the sugar) and a base (on the $1'$ carbon). Nucleotides are bound together to form an RNA molecule or DNA strand by a phosphodiester bond between the $5'$ carbon in the sugar of one nucleotide and the $3'$ carbon in the sugar of the next nucleotide. The phosphate part of the nucleotide forms part of the phosphodiester bond. The sugar and phosphate form the backbone of the RNA molecule or DNA strand whilst the base encodes the genetic information. DNA molecules use four different bases: adenine (A), guanine (G), cytosine (C) or thymine (T). RNA molecules substitute the thymine with another similar base: uracil (U). Due to complementary base pairing adenine is attracted to thymine (or uracil) and guanine is attracted to cytosine. The order of different bases along the backbone is arbitrary.

An RNA molecule consists of a single chain of nucleotides using the sugar β–D–2–ribose. Due to complementary pairing between bases in the RNA molecule short local regions of structure arise. Bases in the chain are attracted and repulsed from one another causing the molecule to take on particular conformations.

RNA, then, has a dual structure: an informational aspect comprising the bases it contains; and a structural aspect due to the folding of the molecule. The structural aspect allows RNA molecules to have a catalytic property. For example, some RNA molecules are able to splice other RNA molecules. Other RNA molecules, such as the rRNA precursor of *Tetrahymena thermophila* are able to splice sequences from themselves using a mechanism called self–splicing (Gardner et al., 1991).

The structure of the DNA molecule was determined by (Watson and Crick, 1953). DNA is predominantly found in cells in a double–stranded form, although it exists in other forms throughout the cell cycle. The double–stranded form consists of two chains (strands) of nucleotides each similar to RNA, but using the sugar β–D–2–deoxyribose instead of ribose. The strands are bound to each other using complementary base pairing and wind into a double helix. Each base in a strand pairs with its complementary base at the same position in the other strand. This means that one strand may be completely determined from the other strand (by taking its complement).

It is important to note that the double–stranded structure of DNA is a completely different concept than diploidy or polyploidy. Polyploidy deals with the number of copies of DNA molecules (or chromosomes). Regardless of the number of copies of chromosomes, each DNA molecule still has two strands.

DNA guides the creation of RNA molecules by acting as a (complementary base pairing) template. This process of reading the bases from one strand of a DNA molecule (called *transcription*) is accomplished using protein and RNA machinery. DNA is read only in the $3'$–$5'$ direction (ie. the $3'$ carbon towards the $5'$ carbon of the deoxyribose sugars) and nucleotides are added to the growing RNA molecule on the $3'$ end. The $3'$–$5'$ direction of a strand is opposite that of the other strand because of the way the strands align themselves, thus strands are read in opposite directions. This is an important subtlety that we include in our model.

DNA is a double–stranded molecule primarily to reduce the number of errors made during replication and transcription. A very low error rate is crucial for organisms because errors limit the number of essential proteins able to be encoded and thus the complexity of the organism.

The double–stranded structure of DNA molecules is significant because (i) it makes DNA more robust and stable than RNA; (ii) it permits the replication of DNA and the required proofreading mechanisms; and (iii) it allows a repair mechanism to operate that uses intact strands as a template for the correction or repair of the associated damaged strand. The repair mecha-

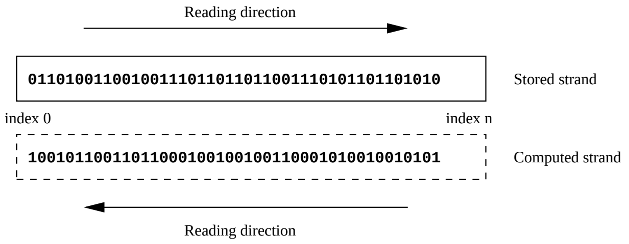


Figure 1: An example of the physical layout of a double–stranded encoding. The double–stranded encoding consists of a simple bit string called the *stored strand*. A second strand (the *computed strand*) is calculated from the stored strand by taking its ones complement. The strands are read in opposite directions.

nism is important because thousands of bases are lost from each DNA molecule each day from spontaneous deamination and depurination caused by thermal fluctuations (Alberts et al., 1994).

These three benefits from double–strandedness are primarily characteristics of the physics of the real world and exist to reduce the errors in genetic information. They do not appear to be useful from a computer science or evolutionary computation perspective — our genomes are sturdy and we need to make an effort to introduce instability and errors rather than the reverse.

3 Model of the Double–Stranded Genome

Our model of a double–stranded genome has been inspired from the structure of the DNA molecule. The two main attributes it models from DNA are (i) the two complementary strands; and (ii) the fact that the strands are read in opposite directions.

We view the commonly used bit string genome from the simple GA as a single strand, much like a molecule of RNA.

A double–stranded encoding (see figure 1) uses a single bit string. This bit string, termed the *stored strand*, represents one of the strands of the DNA molecule. The stored strand is read from bit 0 to bit n.

A second strand (called the *computed strand*) is the ones complement of the stored strand. This strand is read in the opposite direction than the stored strand, starting at bit n through to bit 0. As the computed strand is completely determined from the stored strand it need not be physically stored.

The total genome, therefore, allows access to twice the number of bits than are actually stored. Bits are read starting from index 0 of the stored strand through to index n, then from index n of the computed strand back to index 0. Thus, all stored bits are used twice.

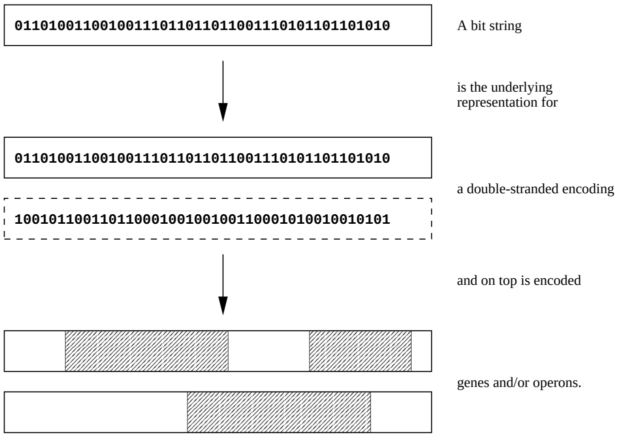


Figure 2: Levels of structure within the double–stranded encoding. At the lowest level the encoding uses a simple bit string. From this bit string we generate two strands. Onto this physical storage we overlay a logical encoding using some language of genes or operons.

Logical genes are encoded onto the raw bit strands using a language (see figure 2). The language should use blocks of contiguous bits as tokens. In the experiments reported in this paper we employed blocks of four bits (nibbles) and a strand length of 400 bits. Particular details of the language used to encode genes are unimportant and depend closely on the problem being solved.

However, any genomic language used should meet two conditions for it to be effective: (i) the semantics of a gene in the language must not be a function of its loci; and (ii) the language should ensure that the genome contains non–coding areas.

Constraints between (complementary) genes on the stored and computed strands arise during evolution. These constraints degrade the performance of the double–stranded encoding. The two conditions above ensure that these constraints are kept to a minimum.

In the experiments with double–stranded encodings in this paper we used an operon expression language which permitted regulation and expression of genes (Kennedy and Osborn, 2000b). This language occasioned a complex dynamic phenotype, but the mechanics of double–stranded encodings are not dependent on such a complex language and phenotype. It is entirely possible to employ a simpler language that just specifies the genes (ie. with no regulation or operons) for a static (or nonexistent) phenotype.

Each bit in the bit string representing a double–stranded encoding is read twice, once as itself and once as its complement. The bit will most probably have a different meaning on each strand because it is read in the reverse direction and has a different context because of the surrounding bits (which are also complemented and read in the reverse). For example, suppose that the stored strand contains the bit string "1101" and that this bit pattern represents the `<start gene>` token in the genomic language. The same bits on the computed strand are read as "0100" which may represent a different token in the language.

It is possible, then, for bits from the raw bit string to participate in up to two genes simultaneously and to have up to two different meanings. We say *up to two genes* because the bits may be part of a non–coding region of the genome.

The fact that bits from the raw bit string may participate in two genes simultaneously is alluring but there is a possible drawback: constraints between the strands arise during evolution as selection tries not only to solve the problem posed but also to find bit patterns that encode genes on both strands.

4 Genetic Operators with Double–Stranded Encodings

The standard genetic operators of crossover and mutation are still applicable to double–stranded encodings. As well, the two strands in the double–stranded encoding permit a simple biologically plausible formulation of the inversion operator.

In single–stranded encodings like those often used in the simple GA, mutation is modelled by flipping a bit randomly with low probability. A similar scheme is employed with double–stranded encodings. With small probability a bit on the stored strand is flipped. When this occurs the corresponding bit on the computed strand is also changed: the computed strand always remains the complement of the stored strand. Every mutation event affects two bits, so the mutation rate should be halved to compensate. In the experiments in this paper we set the mutation rate at 0.005 which mutated 2 bits in each strand on average.

Any recombination algorithm that may be applied to simple bit strings (single–stranded encodings) may be used with double–strand encodings. Crossover is applied to the stored strand of two parent genomes as normal. Once the stored strands have been recombined, the computed strand for each individual is determined. As with mutation, the crossover rate may be reduced to take account of the fact that two strands are affected instead of one.

In our experiments we used random numbers generated from an exponential distribution (parameterised on the crossover rate) to find the distances between successive crossover points on the strand and then crossed over between these points. This algorithm has a similar effect to two–point crossover, but more points are possible. Since we read bits from our genomes in blocks of four, we force crossover to occur only between block boundaries. We chose a crossover rate that yielded on average four crossover points per genome.

In our encoding scheme, where genes are not forced to begin at prescribed loci, we notice that crossover tends to force genes to start at exactly the same index position over a population of individuals. That is, crossover forces alleles to be located at exactly the same locus on the genome.

Inversion is a genetic operator that changes the linkage of genes on the genome. That is, it changes the ordering of the genes. Of course, this can only occur if the semantics of the genes aren't a function of their locus.

The inversion genetic operator has had a chequered past in GA research. It was first described by (Holland, 1992). In the simple GA, the meaning of genes is dependent on their position, so the inversion operator cannot immediately be used. Holland, therefore, modified his encoding scheme to hold pairs of gene index and gene (or bit). The bits could now be permuted without losing their meaning. However, when crossover and inversion were combined, genes could be duplicated or lost entirely from the genome. To avoid this, crossover was constrained to occur only on genomes with genes in the same order.

(Goldberg, 1989) reports that early experiments by Bagley using inversion needed to run for longer because the constraint on the crossover operator effectively forced a much lower crossover rate. No great results from inversion were forthcoming but this may have been because problems were not sufficiently nonlinear (Goldberg, 1989). Inversion has also been applied to ordering problems.

Inversion occurs in cells when both strands of a molecule of DNA break in two places (Gardner et al., 1991). The strands in the broken fragment (between the break points) remain bound together due to the attractions between the complementary base pairs. There are only two ways that the broken fragment may be rejoined: either the same way it was originally connected or rotated through 180° laterally. The latter

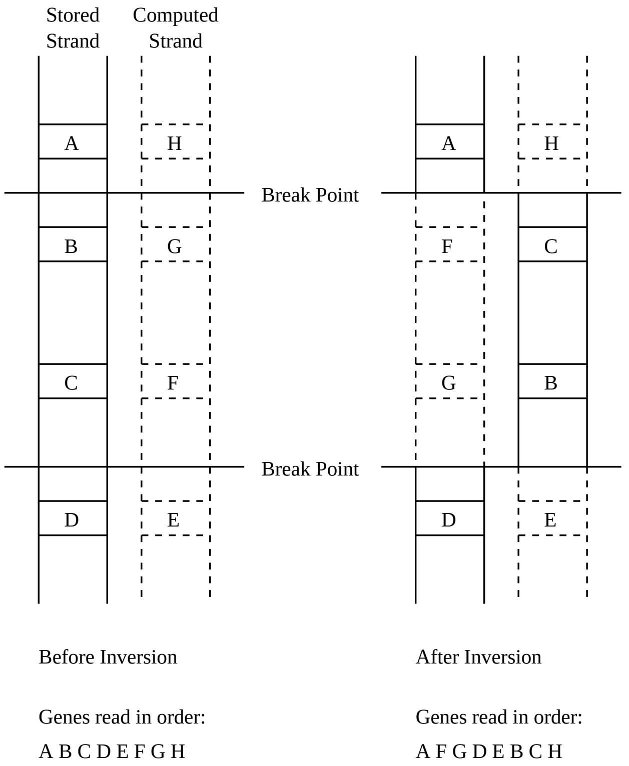


Figure 3: The effect of the inversion operator on the ordering of genes. On the left is the intact double–stranded genome with gene ordering displayed below. On the right the genome has been inverted at the two break points. Genes in the inverted region have moved from one strand to the other and the ordering of the genes has changed as a result.

method of reattachment results in an inversion. It is not possible for the fragment to be rejoined if it spins 180° around the axis parallel to the strands (as opposed to laterally) because the strand needs to "run" in the same direction throughout its entire length (ie. $3'$–$5'$).

Genes wholly in the broken fragment (or inverted region) are moved from one strand to the other. In this way, the order of the genes is changed. Genes that span the break points are disrupted although not necessarily destroyed. Figure 3 illustrates the effect of genes from an inversion.

Formulation of the inversion operator for double–stranded encodings is straightforward. As with the crossover and mutation operators changes are made only to the stored strand but are reflected on the computed strand. Two break points for the strands are chosen uniformly. The bits on the stored strand between these break points are placed in the reverse order. Finally each bit between the break points is replaced with its ones complement. The computed strand can then be read. Figure 4 illustrates the algorithm.

	1	3	4	7	B	9	
Before Inversion	0001	0011	0100	0111	1011	1001	stored strand
	1110	1100	1011	1000	0100	0110	computed
	7	3	D	1	2	6	
			break points				
Reverse	0001	1101	1110	0010	1100	1001	
Negate	0001	0010	0001	1101	0011	1001	
	1	2	1	D	3	9	
After Inversion	0001	0010	0001	1101	0011	1001	
	1110	1101	1110	0010	1100	0110	
	7	B	7	4	3	6	

Figure 4: The action of the inversion algorithm as applied to a short sample double–stranded encoding. At the top is shown the two strands before inversion and the value of each four bit block. Below this, the region of the stored strand between the break points is reversed. Next the complement of the bits is taken. At the bottom the final state of the genome is shown.

As with crossover, we ensure that break points lie on the four–bit boundaries (of our implementation of the inversion operator) so that the token blocks are moved rather than moved and disrupted. This, however, depends on the encoding used.

5 Schemata and Double–Stranded Encoding

It is interesting to examine the relationship between double–stranded encodings and schemata. GAs search using schemata (Holland, 1992). A genome provides sample values for the fitness for all the schemata it represents.

For a single–stranded encoding of length n with each detector able to take on a attribute values from the set V, there are a^n possible encodings and $(a+1)^n$ schemata. Each string in the single–stranded encoding represents (ie. samples the fitness of) 2^n schemata.

Consider a similar double–stranded encoding scheme with strands of length n. Again, each detector in the strands may take on one of a attribute values. A complementary pairing must exist between members of V so that the attribute value on the computed strand may be determined. The complementary pairing should map all values from V back into the same set. For the hypothetical double–stranded encoding there are also a^n distinct genomes. This is because the entire genome is determined only from the stored strand.

There are, however, more schemata associated with this encoding. The number of schemata associated with the stored strand is the same as for single–stranded encodings. Schemata values for the computed strand, though, may be either the complement of the matching value on the stored strand or the "don't care" value. The number of schemata, then, for double–stranded encodings is $(a+1)^n\, 2^n$.

The number of schemata represented by one double–stranded genome is calculated similarly. Positions on both strands may take the "don't care" value or the actual attribute value that is on the strand. So $2^n 2^n$, or 2^{2n}, schemata are represented by a double–stranded genome.

Table 1 summarises this information and compares the numbers of schemata between single–stranded encodings, double–stranded encodings and single–stranded encodings of twice the length.

The effect of the genetic operators on schemata for double–stranded encodings is a little more complicated than for single–stranded encodings. This is because each operator affects two strands at the same time.

Mutation in single–stranded encodings disrupts schemata. With a double–stranded encoding, however, two bits are changed at the same time.

In single–stranded encodings, crossover makes new instances of schemata already in the population and generates new schemata that have not yet occurred. A similar effect occurs with double–stranded encodings but the effect again is amplified because the crossover event modifies two strands at the same time.

Inversion in both single–stranded and double–stranded encodings changes schemata by reordering their parts. As with crossover, longer or higher order schemata are more at risk from disruption.

As described in section 3 we read blocks of four bits from the genome at a time in our experiments. These four bit blocks (or nibbles) permit us to encode up to 16 different tokens in the genomic language. The token coding we use, however, is redundant and there are only 14 tokens in our language. This redundancy is analogous to that in the genetic code between codon and amino acid. We make the biological analogy of a nibble representing a codon and a token standing for an amino acid.

Table 2 compares the numbers of schemata associated with two single–stranded encodings and two double–stranded encodings. We used the redundant 4–bit encoding described above with length eight as well as a binary encoding of length 30. These particular lengths were selected because they permit approximately the same numbers of genomes (10^9). It is clear from the table that the number of schemata associated with double–stranded representations is far greater than for single–stranded encodings containing the same number of bits. Although the number of genomes able to be represented is the same for both single–stranded and double–stranded encodings, the number of schemata that each double–stranded genome represents is much greater than for single–stranded encodings. This implies that each double–stranded genome provides fitness samples for more schemata and thus searches more widely than the single–stranded encoding with the same number of bits.

(Holland, 1992) suggests that genetic algorithms will work better with genomes with many detectors having only a few values (eg. a bit string) than with genomes with fewer detectors each having many values (for example, our nibble genome). Holland's reasoning for this is that, for encodings with a similar cardinality of genome space, the encoding scheme expressing the most schema will search genome space the most efficiently. This same argument explains why the double–stranded encoding can search more efficiently than a single–stranded one.

A single–stranded encoding with twice the number of bits should search even more efficiently because it allows more schemata. However, we are interested in comparing encoding schemes based on the same number of stored bits.

The schemata themselves do not tell the whole story here, though. As described in section 3 the fact that the computed strand is derived from the stored strand means that constraints are set up between genes on either side of the encoding. That is why it is important for the genomic language to allow parts of the genome that do not encode anything. These regions relax the constraints felt by genes on the complementary bases on the other strand.

6 Comparing Single–Stranded and Double–Stranded Encodings

We wished to compare the single–stranded and double–stranded encodings empirically. As mentioned in section 1 we developed the double–stranded encoding as part of a simulation of a biological cell. The

Table 1: Numbers of genomes and schemata that may be represented for three encoding schemes as well as the number of schemata that a genome from the encoding represents. The parameter n is the length of the encoding (number of detectors) and a is the number of attribute values each detector may take.

Encoding scheme	Number of genomes	Number of schemata	Number of schemata that each genome represents
Single–stranded length n	a^n	$(a+1)^n$	2^n
Single–stranded length $2n$	a^{2n}	$(a+1)^{2n}$	2^{2n}
Double–stranded length n	a^n	$(a+1)^n\, 2^n$	2^{2n}

Table 2: Comparative numbers of genomes and schemata associated with four different encoding schemes.

Encoding scheme	Number of genomes	Number of schemata	Number of schemata that each genome represents
Single–strand binary	$2^{30} \cong 1 \times 10^9$	$3^{30} \cong 2.06 \times 10^{14}$	$2^{30} \cong 1 \times 10^9$
Single–strand nibble	$14^8 \cong 1 \times 10^9$	$15^8 \cong 2.56 \times 10^9$	$2^8 \cong 2.56 \times 10^2$
Double–strand binary	$2^{30} \cong 1 \times 10^9$	$6^{30} \cong 2.21 \times 10^{23}$	$2^{60} \cong 1.15 \times 10^{18}$
Double–strand nibble	$14^8 \cong 1 \times 10^9$	$30^8 \cong 6.56 \times 10^{11}$	$2^{16} \cong 6.55 \times 10^4$

experiment comparing single–stranded and double–stranded encodings was performed using this simulation. The actual problem we solved was: could we evolve a simulation of a single–celled organism that could survive in a particular environment? This was called the "bootstrapping problem". For a detailed description of the cell simulation, the bootstrapping problem and the fitness function used see (Kennedy, 1998). The bootstrapping problem could be solved using both representations.

A single–stranded experiment was conducted with genomes the same length (400 bits) as one of the strands in the double–stranded experiment. A second experiment with single–stranded encodings used a genome double this length. The double–stranded experiment used two strands of 400 bits each.

Since the effect of the genetic operators on single–stranded encodings is different to double–stranded encodings, it is difficult to create an experiment that fairly compares single–stranded encodings with double–stranded encodings. As a first approximation, we doubled the mutation and crossover rates for the single–stranded experiments. The inversion operator was not permitted for the double–stranded experiment, as we did not implement an analogue in the single–stranded coding.

The fitness function and all other parameters apart from crossover rate, mutation rate, encoding scheme and strand length were the same for all experiments.

Figure 5 plots the average fitness and maximum fitness for these experiments. Although these results show only one run, we ran many trials and these results are representative.

An explanation is necessary for the decreases in average and maximum fitness values. This is a characteristic of the bootstrapping problem. The GA finds genomes that provide the cell simulation *strategies* for living in the environment. Strategies may be fit initially but go "sour" over a longer time as the cell changes. Because cells are "frozen" when their simulated time ends and because a form of elitism is used, such strategies produce fit parents that issue weak progeny. The maximum fitness decreases when such individuals eventually leave the population.

Initially, the average fitness for the double–stranded experiment is higher and grows faster than either of the single–stranded experiments. Eventually, however, the average fitness values of the single–stranded experiments outstrip the double–stranded genomes. The increase in average fitness of the second single–stranded experiment (the one with genome of length 800 bits) is much slower than the first single–stranded experiment.

The initial fast increase in average fitness of the double–stranded experiment suggests that the increase in schemata from the double–strand is valuable at the beginning when the GA is exploring many options. Later, however, the strong relationship between the strands causes the GA to stall because of the extra

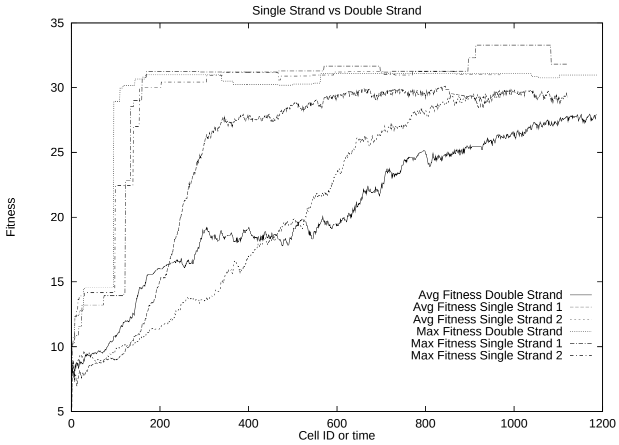


Figure 5: Results from an experiment comparing single–stranded genomes to double–stranded genomes. Values for average and maximum fitness are shown for each kind of genome. The single–strand 1 lines represent a single–stranded genome of 400 bits, single–strand 2 is a genome with 800 bits and double–strand is a double–stranded encoding with 400 bits in each strand.

constraints that need to be optimised. A beneficial mutation on one strand, for example, may cause a destructive mutation on the other strand. The GA must perform its optimisation with the constraint imposed by the relationship between the strands.

This effect is similar to that described by (Kauffman, 1993) in his NK model of rugged fitness landscapes. Kauffman models genomes consisting of N genes. The fitness contribution of each gene depends on itself and K other genes on the genome. Evolution of genomes comprises selection and mutation of genes with no recombination. As K, a measure of the epistatic interactions, increases the fitness landscape becomes more rugged. When K is 0, the landscape consists of one global peak. At the other end of the scale, where K is $N - 1$, the landscape is fully random, the number of local optima is extremely large and the height of the fitness peaks decreases towards the average fitness. As the number of interactions between genes increases, Kauffman says that "optimisation can attain only ever poorer compromises". This is exactly the problem we face. Relationships between genes on single–stranded genomes are those imposed by the genomic language and the problem itself. Double–stranded genomes, however, add the additional constraint of the complementary matching between strands.

Toward the start of the runs the experiment using the 800 bit single–strand (single strand 2) finds higher maximum fitness values before the shorter single–strand. However the shorter single–strand experiment eventually finds individuals with higher fitness. The double–strand experiment also finds fit individuals early. This seems to support the relative numbers of schemata associated with genomes of each type.

However, the average fitness of the single–stranded encoding with the longer strand (single strand 2) climbs slower than that of the shorter strand (single strand 1). This suggests that although fit individuals are found, the population as a whole cannot converge onto them. There may be three reasons for this effect: (i) these fit individuals may encode strategies with short term benefits (in which case we would expect to see maximum fitness values decrease as they are flushed out of the population); (ii) the longer genomes are more susceptible to breakage from mutation and crossover; or (iii) a longer genome has the room to create more genes / strategies (most of which are poor).

Runs were also made (but not shown here) with mutation and crossover rates for the single–stranded experiments the same as for the double–stranded experiment (ie. halved). The single–strand experiments performed even better then.

7 Varying the Rate of Inversion

Our next set of experiments focussed on the inversion operator. We ran experiments using different rates of inversion. All other parameters were the same for each experiment. As in the previous experiments cell simulations were evolved to live in a simple environment.

Three experiments were conducted with inversion rates 0, 0.15 and 0.3. Each rate is the probability that a genome is inverted. A rate of 0 means that inversion does not occur. Double–stranded genomes, of course, were used in the experiments.

Figure 6 graphs the average and maximum fitnesses for these experiments. Although these results show only one run, we ran many trials and these results are representative.

The experiments performed with the "bootstrapping problem" show us that the problem is characterised by high fitness found early but then not much improved upon (except perhaps for strategies that seem to have short term gain but longer term problems ie. where the maximum fitness reduces again).

The inversion experiments show two main results: (i) within the framework of the bootstrapping problem, the maximum fitness values are found earlier when using higher rates of inversion; and (ii) the experiments with higher rates of inversion produced higher average

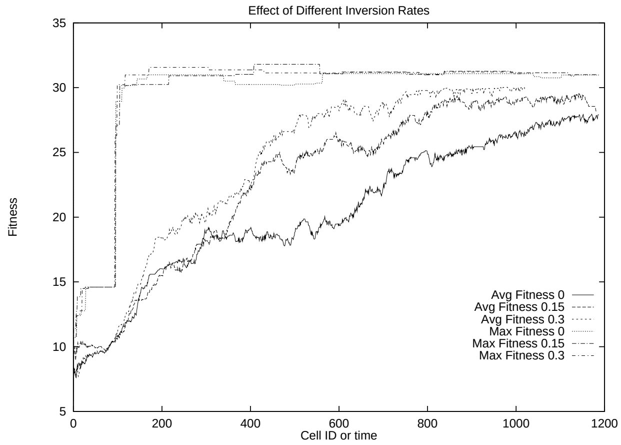


Figure 6: Graph showing an experiment using different rates of inversion.

fitnesses meaning that convergence to the maximum fitnesses was faster with more inversion.

Inversion permutes the genome and favours shorter genes (since longer ones are disrupted). It is not immediately clear, though, why this should increase the convergence of the population without significantly increasing the maximum fitness. Perhaps the maximum fitness has such a small region to increase into that it simply isn't discernible. This experiment has produced favourable results, but more favourable experiments using a harder problem with more stages to finding the maximum fitness are necessary for an explanation.

8 Conclusions

This paper presents a new biologically inspired encoding scheme based on the double–stranded DNA molecule. The double–stranded encoding scheme boosts the number of schemata that may be represented with a given number of bits, thus increasing the breadth and efficiency of search. However, the double–stranded encoding scheme is a double edged sword. Whilst the number of schemata searched is larger, constraints emerge between the strands that add to the complexity of the problem being solved.

The double–stranded encoding scheme offers two advantages over a single–stranded encoding: (i) the number of schemata that are represented by each genome is much larger; and (ii) a biologically inspired algorithm for the inversion genetic operator is simple to implement. The increase in number of schemata for double–strand encodings gives the genetic algorithm more information to use and allows faster search of the problem space. Greater rates of inversion were shown to increase the convergence of a population, but the reasons for this remain unclear.

We hypothesise that the major drawback of using a double–strand encoding scheme is that the strands impose a further constraint on the genetic algorithm: modification of information on one strand may adversely affect information on the other strand. The constraints decrease the rate of convergence of a population.

9 Future Work

Further research with other problems, in particular standard test problems, is required to determine whether the hypothesis mentioned in the last section stands. We need to use double–stranded encodings in problems that are not evolving *strategies*. That is, where the fitness is not a moving target and where the unfortunate situation of highly fit solutions producing poor offspring does not exist.

If, indeed, the inter–strand constraints are a problem, a variety of approaches to decrease this effect present themselves: (i) a higher rate of inversion; and/or (ii) a sparser density of genes (perhaps caused by a different genomic language or longer genomes). Another technique may help: use the double–stranded encoding only at the start of evolution where the increased breadth of search is useful. After some time, fold the strands out into a longer single–stranded encoding to eliminate the inter–strand constraints.

The single–strand experiments always seem to perform better than the double–strand experiment, but DNA is a double–stranded coding. Why should biology use an inferior solution? Surely the answer lies in the other benefit of double–stranded codings, the one that has no analogue in our model. That is, that two strands ensure greater stability of the DNA molecule and reduce the error rate. Perhaps the interconnectedness of the strands is the price biology pays to build complex organisms.

References

Alberts, Bray, et al. (1994). *Molecular Biology of the Cell.* Garland Publishing, New York, third edition.

Gardner, E. J., Simmons, M. J., and Snustad, D. P. (1991). *Principles of Genetics.* John Wiley & Sons, eighth edition.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison–Wesley.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, Massachusetts, first MIT press edition.

Kargupta, H. (2000). The genetic code and the genome representation. In (Wu, 2000), pages 179–184.

Kauffman, S. A. (1993). *The Origins of Order: self-organization and selection in evolution.* Oxford University Press, New York.

Kennedy, P. J. (1998). *Simulation of the Evolution of Single Celled Organisms with Genome, Metabolism, and Time–Varying Phenotype.* PhD thesis, University of Technology, Sydney.

Kennedy, P. J. and Osborn, T. R. (2000a). Evolution of adaptive behaviour in a simulated single–celled organism. In Meyer, J. A., Berthoz, A., Floreano, D., Roitblat, H., and Wilson, S., (Eds.), *SAB2000 Proceedings Supplement Book*, pages 225–234, Honolulu. International Society for Adaptive Behaviour.

Kennedy, P. J. and Osborn, T. R. (2000b). Operon expression and regulation with spiders. In (Wu, 2000), pages 161–166.

O'Neill, M. and Ryan, C. (2000). Incorporating gene expression models into evolutionary algorithms. In (Wu, 2000), pages 167–172.

Rosenberg, R. S. (1967). *Simulation of Genetic Populations with Biochemical Properties.* PhD thesis, University of Michigan.

Simões, A. and Costa, E. (2000). Using genetic algoriths with asexual transposition. In Whitley, D., Goldberg, D., Cantú-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 323–330, San Francisco. Morgan Kauffmann.

Watson, J. D. and Crick, F. H. C. (1953). Molecular structure of nucleic acids. a structure for deoxyribose nucleic acid. *Nature*, (171):737–738.

Weinberg, R. (1970). *Computer simulation of a Living Cell.* PhD thesis, University of Michigan.

Wu, A. S., (Ed.) (2000). *Proceedings of The 2000 Genetic and Evolutionary Computation Conference Workshop Program, Las Vegas, Nevada, 8 July 2000.*

A Hybrid Genetic Search for Multi-way Graph Partitioning Based on Direct Partitioning

Jong-Pil Kim and Byung-Ro Moon
School of Computer Science & Engineering
Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{jpkim, moon}@soar.snu.ac.kr

Abstract

Multi-way partitioning is an important extension of two-way partitioning as it provides a natural and direct model for many partitioning applications. In this paper, we propose a hybrid genetic algorithm for k-way partitioning. The algorithm includes an efficient local optimization heuristic which is not based on the bisection method. It considerably outperformed existing k-way partitioning algorithms.

1 Introduction

Partitioning is an important problem that arises in various fields of computer science, such as sparse matrix factorization, VLSI circuit placement, and network partitioning. Good partitioning of a system not only significantly reduces the complexity involved in the design process, but can also improve the timing performance as well as its reliability [2] [14].

Let $G = (V, E)$ be an undirected graph, where V represents the set of vertices and E represents the set of edges. *k-way partition* is a partitioning of the vertex set V into k disjoint subsets. A k-way partition is said to be *balanced* if the difference of cardinalities between the largest and the smallest subsets is at most unity. In this paper, we consider only balanced partition. The *cut size* of a partition is defined to be the number of edges whose endpoints are in different subsets of the partition. The *k-way partitioning problem* is the problem of finding a k-way partition with the minimum cut size. A two-way partition is often called *bisection* or *bipartition*.

Since the k-way partitioning problem is NP-hard [15] even for $k = 2$, and finding good approximation solutions for general graphs or planar graphs is also NP-hard [4], attempts to solve partitioning problems have concentrated on finding heuristics which will yield approximate solutions in polynomial time. Among such methods are the Kernighan-Lin algorithm (KL) [17], the Fiduccia and Mattheyses algorithm (FM) [10], simulated annealing (SA) [15] [18], tabu search (TS) [1] [9] [22], large-step Markov chain (LSMC) [20] [11], and genetic algorithms (GA) [5] [6] [7] [13] [19] [23] [26].

Traditionally most k-way partitioning algorithms have been based on bipartition. Two representative approaches in this context are the pairwise approach and the recursive one. The pairwise approach starts with an arbitrary k-way partition. It picks two subsets at a time from the k subsets and performs bipartitioning to reduce the cut size between the two. A well-known heuristic is the Pairwise KL (PKL), which was suggested in [17]. Figure 1(a) shows the pairwise approach. The other recursively bipartitions the target graph until we have k subsets. A representative heuristic is the Recursive KL (RKL) [24] [25] [5]. Figure 1(b) shows the recursive approach. These heuristics using bipartition for k-way partitioning have several drawbacks. They focus on the current partitions without considering entire partitions, which makes the search rather "local," as shown in Figure 1(a) and Figure 1(b).

Another method is a direct extension of the 2-way FM. Since FM is based on a sequence of single vertex movements, whereas KL is based on a vertex pair swap, FM is more appropriate for extension to k-way partitioning. A k-way partitioning algorithm based on the 2-way FM heuristic is often referred to as k-FM [8]. In the 2-way FM, each vertex can move only to the opposite subset. In the k-way partitioning problem, each vertex has $k-1$ possible destination subsets. k-FM allows movement of any vertex to any of the $k-1$ subsets. This approach enables vertices to move between any arbitrary subsets such that it assists "global" changes in the current configuration, as showed in Figure 1(c).

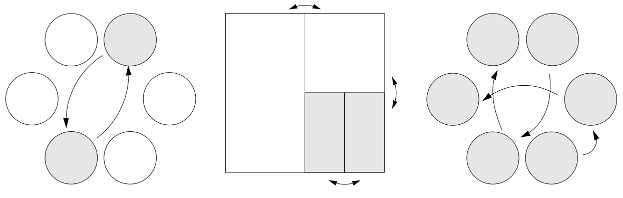

(a) pairwise (b) recursive (c) direct

Figure 1: Three k-way partitioning approaches

Sanchis [25] showed that the direct multi-way partitioning approach obtained better solutions compared to the recursive approach for random networks.

We use a variation of k-FM in our hybrid GA. For conventional k-FM algorithms, each subset has to maintain $k-1$ buckets because each vertex is associated with $k-1$ subsets. This requires $k(k-1)$ buckets and $O(k \cdot N)$ space, where N denotes the total number of vertices. Our algorithm uses one global bucket list. In this case, $O(k \cdot N)$ space is required as usual, but the bucket is maintained like the 2-way FM. We observed that tie-breaking strategies perform differently upon different graphs. We designed the genetic algorithm in such a way that attractive tie-breaking strategies survive through evolution. We also allow the violation of the balance criterion for more flexible search, which also turned out to contribute toward the improvement.

The remainder of this paper is organized as follows. Section 2 summarizes the Fidducia and Matteyses algorithm and provides our local optimization heuristic for the k-way partitioning problem. Detail of the hybrid GA optimization is described in Section 3. We present experimental results in Section 4 and state our conclusions in Section 5.

2 The Heuristic for k-way Partitioning

2.1 Fidducia and Matteyses Algorithm (FM)

The FM algorithm is a local-optimization heuristic for graph bisection. Starting with an initial bisection, FM moves a vertex at a time from one subset to another in an attempt to minimize the cut size.

Let P be one side of a bisection for $G = (V, E)$. For vertices $v \in P$, we denote by g_v the cut size reduction when vertex v moves to the opposite side. The vertex to be moved is chosen on the basis of the balance criterion and its effect on the cut size of the current bisection. If we do not consider the balance criterion, all vertices will eventually migrate to one side except the lowest-degreed vertex. To prevent the vertex-moving process going into an infinite loop, each vertex is locked after movement and can not move any further. Only unlocked vertices are allowed to move. A pass continues choosing vertices and moving them until all vertices are locked or any further move violates the balance criterion. After all moves have been made, the best partition encountered during the pass is taken as the output of the pass.

2.2 Extended FM Algorithm

Sanchis [25] extended the FM algorithm for the k-way partitioning. The algorithm considers all possible moves of each vertex from its home subset to any of the others. It maintains $k(k-1)$ gain bucket lists. We simplified the algorithm to use one global bucket list. Each vertex occupies an entry with the largest gain among the $k-1$ gains. This reduces the running time to find the best move from $k(k-1)$ buckets. Bucket operations such as addition or deletion of an entry occur in the one bucket. So the bucket can be maintained like the 2-way FM algorithm.

Our Extended FM (EFM) algorithm is given in Figure 2. It has two phases, MoveVertex and MakeBalance. In MoveVertex, for each vertex, the algorithm calculates gains for the $k-1$ subsets and stores the best subset with corresponding gain. Then the algorithm selects the vertex which has the highest gain among the vertices whose movements do not violate the balance requirement. After we move the vertex, we adjust gains that are affected by the movement and lock it. If there is no more vertex to move, we undo the sequence of movements at the time that the gainsum is maximal. We continue this until there is no improvement. Then the result of MoveVertex passes into MakeBalance.

The result of MoveVertex may not balance the partition. In this case, we adjust each subset size for the balance in MakeBalance. MakeBalance is similar to Movevertex. The only difference is that MakeBalance focuses on subsets whose sizes are surplus or deficient. MakeBalance selects a vertex from a surplus subset and moves it to a deficient subset. At the end of MakeBalance, the k-way partition satisfies the balance. The nomenclature for the algorithm MoveVertex and MakeBalance is described in Figure 3.

In the MoveVertex of Figure 2, lines 2 through 6 compute the gains for all vertices. Each loop of lines 8 through 17 chooses a vertex, moves and locks it. In lines 18 and 19, we find a sequence that maximizes the gainsum and undoes the remaining sequence of movements to obtain the smallest cut size in a pass. In the MakeBalance, in lines 1 through 5 we find vertices in surplus subsets, calculate the gains of the moves to deficient subsets. Lines 6 through 11 choose a vertex, move and lock it. If every subset satisfies the balance, MakeBalance returns the balanced partition and a pass is terminated.

2.3 Tie-Breaking Strategies

Two kinds of ties occur when the algorithm maintains a global bucket list. First, while the algorithm calculates gains for $k-1$ subsets we may get the same gain for more than one subset. Two tie-breaking strategies were used in this case. One is where we choose a subset which has the smallest index (SMALL). The other is one in which we give preference to the subset to which a vertex moved most recently (RECENT).

There is another type of tie; there may exist more than one vertex in the max gain bucket. Hagen *et al.* [12] observed that the LIFO management of gain buckets yields considerably better solutions than FIFO and Random bucket management for the 2-way FM heuristic. We, however, observed that this phenomenon is not consistent in our multi-way partitioning.

In the bipartition with LIFO management, neighbors of the recently moved vertices have a higher chance of winning a tie break. Hence LIFO management deals with clusters effectively. However, this is not the case in LIFO management for the k-way partitioning problem when k is greater than 2. Since there are $k-1$ candidate subsets for moving, it is not easy to concentrate an one or two subsets. Moreover, an arbitrary subset has a limit in inhaling a cluster unless some vertices do not move out.

We combined Random and LIFO management. First, we try to use LIFO management. When the last vertex moved to the subset P, and the vertex at the front of the max gain bucket is ready to move to P, we move the vertex to P regardless of the balance criterion in an attempt to move the vertex along with its neighbors. If the front vertex is not headed for P, we use the Random management. we call this tie-breaking rule Combined management. We examined the performance of LIFO, Random, and Combined management. The experimental results are in Table 1. One can observe that Combined management overall shows better results than LIFO and Random bucket management.

Hereafter, we refer to SMALL+Combined management as EFMA and RECENT+Combined management as EFMB.

```
Extended FM ($G$, $P$)
        // $P$: a given initial subset;
        repeat {
            $P \leftarrow$ MoveVertex($G$, $P$);
            $P \leftarrow$ MakeBalance($G$, $P$);
        } until (stopping condition)

MoveVertex($G$, $P$)
1.  do {
2.      for each $v \in V$ {
3.          $\forall i = 1, 2, \ldots, k$, calculate $g_v[i]$;
4.          $g_v \leftarrow \max_{1 \le i \le k} g_v[i]$;
5.          add $v$ to $FreeList$;
6.      }
7.      $i \leftarrow 0$
8.      do {
9.          Choose a vertex $v \in FreeList$ s.t.
                                    $g_v$ is maximal;
10.         Find $a \in \{1, 2, \ldots, k\}$ that maximizes $g_v[a]$;
11.         Move $v$ to subset $a$ and lock $v$;
12.         Adjust gains that are affected by
                                    $v$'s movement;
13.         Remove $v$ from $FreeList$;
14.         $i$++;
15.         $\xi_i \leftarrow g_v$;
16.     } until ( $FreeList$ is empty )
17.     Find $l \in \{1, \ldots, i\}$ that maximizes $\sum_{j=1}^{l} \xi_j$;
18.     undo the sequence of movements from $l+1$ to $i$;
19. } until ( There is no improvement )

MakeBalance($G$, $P$)
1.  for each $v \in \bigcup\{ V_i \mid |V_i| > |V_i|_{org}\}$ {
2.      $\forall j \in \{ i \mid |V_i| < |V_i|_{org}\}$, calculate $g_v[j]$;
3.      $g_v \leftarrow \max_j g_v[j]$;
4.      add $v$ to $FreeList2$;
5.  }
6.  do {
7.      Choose a vertex $v \in FreeList2$ s.t.
                                $g_v$ is maximal;
8.      Find $a \in \{1, 2, \ldots, k\}$ that maximizes $g_v[a]$;
9.      Move $v$ to subset $a$ and lock $v$;
10.     Adjust gains that are affected by $v$'s movement;
11. } until ( Every subset satisfies the balance )
```

Figure 2: Extended FM Algorithm

V_i : subset i, $\bigcup_{1 \le i \le k} V_i = V$
$|V_i|_{org}$: original size of V_i
$g_v[i]$: the gain of moving vertex v to subset i
g_v : the maximum among $g_v[i]$'s for all $i = 1, \ldots, k$
ξ_i : the gain of i^{th} movement in MoveVertex

Figure 3: The nomenclature for the EFM

```
Create initial population of fixed size;
do {
    Choose parent1 and parent2 from population;
    normalization(parent1,parent2);
    offspring ← crossover (parent1,parent2);
    EFMA(offspring) or EFMB(offspring)
                    according to the probability;
    replace (population,offspring);
} until (stopping condition);
return the best solution;
```

Figure 4: The Genetic Extended FM algorithm for multiway partitioning (GEFM)

2.4 Time Complexity

The FM algorithm uses a bucket list structure to maintain the gains of vertices. The complexity of the FM algorithm is $O(|P|)$ where P is the number of pins in a circuit graph (hypergraph). Since the graphs used in our experiments do not have hyperedges (edges with more than two endpoints), the time complexity of the FM algorithm is $O(|E|)$.

The time complexity of the Sanchis algorithm [25] is $O(k \cdot |E| \cdot l \cdot (\log k + lp))$ where p is the maxdegree of the graph. l comes from the concept of "level gain." If we ignore the concept of level gain, the time complexity is $O(k \cdot |E| \cdot \log k)$. The fact that the Sanchis algorithm maintains $k(k-1)$ bucket lists causes additional load in the processing. The term $\log k$ comes from the maintenance of the heap for keeping the max gain vertex. In our algorithm, we easily find a max gain vertex in the one bucket. Hence the term $\log k$ can be removed and our implementation of k-FM takes $O(k \cdot |E|)$.

3 Hybrid Genetic Algorithm

A genetic algorithm starts with a set of initial solutions (chromosome) which are called a population. This population then evolves into different populations over a number of iterations. At the end the best member of the population is returned by the algorithm as the solution to the problem. If we add a local improvement heuristic, typically after mutation, we say it is hybridized and a GA with this scheme is called a hybrid GA. Our hybrid genetic algorithm is described in Figure 4. We used the general structure of hybrid steady-state genetic algorithms. In the following, we denote the framework by GEFM.

- *Encoding*: A k-ary string for each chromosome to represent a k-way partition is used. For example, if the i^{th} vertex belongs to subset j, the value of the i^{th} gene is j.
- *Initialization*: First, p chromosomes are created. Each chromosome keeps a balanced partition scheme. We set the population size p to be 50 in GEFM.
- *Selection*: The roulette-wheel-based *proportional selection* scheme is used. The probability that the best chromosome is chosen was set to four times higher than the probability that the worst chromosome is chosen.
- *Crossover and Mutation*: We normalized the parents before crossover. This was done in [19] and helps maintain consistency between the two parents. We used a five-point crossover operator. After crossover, chromosomes are usually not balanced. We start at a random point on the chromosome and adjust the gene values to the right until the balance is satisfied. This has some mutation effect, so we do not add any specific mutation.
- *Local Optimization*: Since the merits of EFMA and EFMB for the k-way partitiong problem can differ according to the graph, we used EFMA and EFMB together. At the begining, one of EFMA and EFMB is chosen as a local optimization heuristic with identical probability by the genetic algorithm. If EFMA is chosen and the offspring tuned by EFMA replaces a solution tuned by EFMB, we increase the probability that EFMA is chosen. The opposite case for EFMB is handled in the same manner. Formally, the probability that EFMA is chosen is $A/A+B$ where A and B are the numbers of solutions tuned by EFMA and EFMB, respectively. In fact, $A+B$ is the size of the population. In this way, we are relieved of the choice of an appropriate local optimization heuristic depending on different graphs.
- *Replacement*: If it is superior to the closer parent in Hamming distance, the offspring replaces the closer parent, if not,the other parent is replaced if the offspring is better. If not again, the worst in the population is replaced.
- *Stopping Condition*: For stopping, we use the number of consecutive fails to replace one of the parents. We set this number to be 50 in GEFM.

4 Experimental Results

4.1 The Test Set and Test Environment

Before showing the experimental results, we first introduce the benchmarks used in this experiment. The

Table 1: Cut sizes of Experiments with tie-breaking strategies

Graph	**LIFO**		**Random**		**Combined**	
	Best[1]	Average[2]	Best[3]	Average[4]	Best[5]	Average[6]
G500.20	4080	4109.78	4077	4105.97	4066	4091.54
G1000.2.5	349	366.25	352	373.73	343	360.23
U500.40	5381	5518.51	5362	5478.48	5391	5474.34
U1000.05	193	240.87	213	257.14	164	203.32
reg500.20	230	240.94	226	239.61	228	239.94
reg5000.0	2004	2068.62	2016	2082.61	1929	2017.65
cat.5252	565	670.24	632	729.45	593	677.94
rcat.134	91	92.23	91	92.50	91	92.42
grid5000.50	751	873.05	792	961.80	702	750.60
w-grid100.10	128	129.49	128	129.20	128	130.03

1. The best cut size of 1,000 runs with LIFO bucket management
2. The average cut size of 1,000 runs with LIFO bucket management
3. The best cut size of 1,000 runs with Random bucket management
4. The average cut size of 1,000 runs with Random bucket management
5. The best cut size of 1,000 runs with Combined bucket management
6. The average cut size of 1,000 runs with Combined bucket management

graphs are composed of eight random graphs and eight geometric graphs with the number of vertices ranging from 500 to 1,000, eight random regular graphs, eight caterpillar graphs, and eight grid graphs with the number of vertices from 134 to 5,252. They are described in [15] [5] [3]. These graphs have also been used in a number of other papers on graph partitioning [1] [26] [21]. The different classes are briefly described below.

1. G$n.d$: A random graph on n vertices, where an edge is placed between any two vertices with probability p chosen so that the expected vertex degree, $p(n-1)$, is d.

2. Un, d : A random geometric graph on n vertices that lie in the unit square and whose coordinates are chosen uniformly from the unit interval. There is an edge between two vertices if their Euclidean distance is t or less, where $d = n\pi t^2$ is the expected vertex degree.

3. reg$n.b$: A random regular graph on n vertices each of which has degree 3, and for which the optimal bisection size is b with probability $1-o(1)$, see [3].

4. cat.n : A caterpillar graph on n vertices, with each vertex having six legs. It is constructed by starting with a straight line (called the spine), where each vertex has degree two except for the outermost vertices. Each vertex on the spine is then connected to six new vertices, the legs of the caterpillar. With an even number of vertices on the spine, the optimal bisection size is 1. rcat.n is a caterpillar graph with n vertices, where each vertex on the spine has $\sqrt{n}$ legs. All caterpillar graphs used here have an optimal bisection size of 1.

5. grid$n.b$: A grid graph on n vertices and whose optimal bisection size is known to be b. w-grid$n.b$ denotes the same grid but the boundaries are wrapped around.

We conduced tests on 32-way partitioning. A C language program was used on a Pentium III 866MHz computer with Linux operating system. It was compiled using GNU's *gcc* compiler.

4.2 Experimental Results

We first examine the performance of the suggested local optimization heuristics, EFMA and EFMB, against the well-known partitioner MeTis [16]. Then we show the experimental results of the hybrid GA. Since the hybrid GA uses the local optimization heuristic as an engine, it is obvious that the hybrid GA would perform better than the local optimization heuristic. One may ask how the genetic process affected the performance. We thus examine the effectiveness of genetic search by comparison with multi-start local optimizations with comparable time budgets.

Table 2 shows the results on 32-way partitioning. EFMA, EFMB, and MeTis are compared. The statistics of EFMA and EFMB are from 1,000 independent runs; so the average results are fairly stable. Since MeTis is a deterministic algorithm, it is not possible to obtain more than one solution; thus, each result of MeTis is from a single run. The bold-faced numbers indicate the best average result from among the three algorithms. MeTis is much faster than the other algorithms. On average, EFMB performed best; however, for some graphs, MeTis had the highest performance.

The genetic algorithm(GEFM) significantly improved the performance of EFMA and EFMB. However,

Table 2: The Results of 32-way Partitioning

Graph	**MeTis**		**EFMA**			**EFMB**		
	Average[1]	CPU[2]	Best[3]	Average[3]	CPU[2]	Best[3]	Average[3]	CPU[2]
G500.2.5	205	0.02	187	199.78	0.29	185	**195.63**	0.37
G500.05	695	0.02	643	661.08	0.45	640	**654.52**	0.55
G500.10	1675	0.02	1602	1619.56	0.77	1597	**1616.37**	0.94
G500.20	4189	0.06	4065	4094.65	1.73	4066	**4091.54**	2.03
G1000.2.5	363	0.02	348	368.31	0.83	343	**360.23**	0.91
G1000.05	1318	0.03	1262	1297.54	1.44	1243	**1277.35**	1.57
G1000.10	3558	0.05	3426	3457.37	2.45	3408	**3446.12**	2.98
G1000.20	8035	0.15	7898	7944.29	4.81	7886	**7936.50**	5.63
U500.05	211	0.01	121	**141.39**	0.37	121	142.68	0.54
U500.10	697	0.03	540	572.53	0.74	544	**571.73**	0.87
U500.20	2060	0.01	1841	**1893.49**	1.55	1848	1894.81	1.58
U500.40	5752	0.03	5378	5476.21	2.69	5391	**5474.34**	2.88
U1000.05	209	0.01	150	**194.90**	0.86	164	203.32	1.29
U1000.10	745	0.04	614	693.36	1.81	605	**688.80**	2.20
U1000.20	2761	0.05	2428	2512.68	3.65	2425	**2511.51**	3.91
U1000.40	8314	0.09	7443	**7590.90**	7.03	7437	7593.25	7.25
reg500.0	247	0.01	223	240.54	0.38	221	**234.34**	0.39
reg500.12	248	0.01	225	238.39	0.37	221	**233.11**	0.38
reg500.16	251	0.03	225	240.88	0.37	223	**235.46**	0.38
reg500.20	272	0.01	232	246.29	0.38	228	**239.94**	0.38
reg5000.0	**1878**	0.09	2044	2133.55	54.26	1929	2017.65	13.07
reg5000.4	**1861**	0.09	2033	2136.11	53.79	1929	2019.07	13.33
reg5000.8	**1883**	0.08	2041	2140.41	53.72	1951	2022.45	13.21
reg5000.16	**1848**	0.10	2034	2134.30	54.31	1942	2018.52	13.42
cat.352	**87**	0.01	85	87.23	0.16	85	94.63	0.20
cat.702	**68**	0.01	59	76.75	0.35	86	112.15	0.56
cat.1052	**92**	0.01	88	104.21	0.75	143	168.39	0.91
cat.5252	**102**	0.09	168	221.90	14.90	593	677.94	15.43
rcat.134	95	0.01	91	**92.18**	0.04	91	92.42	0.05
rcat.554	178	0.01	159	**162.10**	0.14	159	185.39	0.33
rcat.994	**33**	0.02	31	34.28	0.17	32	246.74	1.45
rcat.5114	**531**	0.15	476	559.36	4.36	687	1038.22	31.53
grid100.10	117	0.01	108	110.56	0.04	108	**110.43**	0.04
grid500.21	263	0.03	223	236.96	0.34	220	**236.44**	0.41
grid1000.20	340	0.03	320	336.33	0.95	317	**335.04**	1.19
grid5000.50	**740**	0.06	700	758.42	9.46	702	750.60	12.03
w-grid100.20	138	0.01	128	130.05	0.04	128	**130.03**	0.05
w-grid500.42	292	0.01	267	280.37	0.36	267	**279.32**	0.39
w-grid1000.40	419	0.04	389	403.16	0.98	388	**401.34**	1.13
w-grid5000.100	911	0.07	850	897.79	10.35	837	**890.26**	11.82

1. From single run
2. CPU seconds on Pentium III 866MHz
3. From 1,000 runs

GEFM took on average 350 times more than a single run of EFMA or EFMB. It is not clear how critical the genetic search is to the performance improvement. Thus, we compared GEFM with multi-start versions which ran local optimization heuristics on 350 random initial solutions and returned the best solution. Table 3 shows the experimental results. The results show that, given a comparable length of time, the genetic search algorithm considerably outperformed the multi-start heuristics.

5 Conclusions

In this paper, we extended the 2-way Fiduccia and Mattheyses algorithm for the multi-way partitioning problem. A distinctive feature of our algorithm is that it used LIFO and Random bucket management in conjunction and it could sometimes violate the balance criterion to obtain better results. By attempting direct k-way partitioning and allowing more freedom to vertex movements, our heuristics improved the solution quality.

The genetic algorithm used both EFMA and EFMB as the local optimization. The merits of these were combined in the GA framework; GA adaptively chooses the local heuristic that is suitable for the current graph in the process of generation. The comparison between multi-start heuristics and GEFM showed the effectiveness of the genetic process.

Our local heuristics can be used in other stochastic methods such as tabu search [22] [9] [1] and large-step Markov chain [20] [11].

Acknowledgements

This research was supported in part by KOSEF through Statistical Research Center for Complex Sys-

Table 3: The Results of 32-way Multi-Start and GEFM

Graph	**EFMA Multi-Start**			**EFMB Multi-Start**			**GEFM**		
	Best[1]	Average[2]	CPU[3]	Best[1]	Average[2]	CPU[3]	Best[4]	Average[4]	CPU[3]
G500.2.5	184	188.02	103.64	184	185.72	143.49	180	184.71	98.38
G500.05	640	645.26	159.55	635	640.48	225.30	627	636.39	184.48
G500.10	1593	1600.14	265.95	1593	1597.82	402.38	1574	1586.54	356.45
G500.20	4069	4070.20	606.62	4065	4066.80	420.57	4037	4048.69	872.27
G1000.2.5	338	348.68	302.40	339	342.48	408.74	316	326.97	391.45
G1000.05	1260	1268.50	523.23	1240	1252.46	692.12	1217	1227.93	731.17
G1000.10	3411	3424.52	880.49	3403	3414.32	1182.79	3360	3374.09	1403.28
G1000.20	7902	7905.80	1648.19	7892	7895.60	1848.07	7829	7839.91	2950.90
U500.05	118	122.68	129.26	118	123.00	204.50	115	120.75	156.04
U500.10	539	545.12	260.92	540	545.10	341.00	533	542.09	171.44
U500.20	1833	1846.86	527.99	1839	1847.00	618.10	1834	1845.75	371.32
U500.40	5382	5391.40	928.39	5376	5388.00	955.51	5366	5391.24	742.39
U1000.05	150	159.52	304.17	159	167.08	510.51	132	147.16	615.77
U1000.10	615	625.12	620.92	607	619.20	800.20	578	592.07	1069.63
U1000.20	2407	2430.72	1235.64	2402	2423.68	1355.78	2376	2396.31	1268.35
U1000.40	7420	7437.80	2421.13	7419	7432.40	2445.41	7381	7420.02	1814.45
reg500.0	222	225.92	134.17	220	222.86	137.07	221	226.03	97.46
reg500.12	221	224.90	129.69	220	221.92	134.75	218	222.83	96.11
reg500.16	222	226.98	128.99	220	223.68	133.81	221	224.80	95.09
reg500.20	227	231.72	131.56	224	228.24	133.90	225	231.53	96.01
reg5000.0	2017	2042.10	13473.76	1922	1941.83	5464.18	1690	1724.71	7630.20
reg5000.4	2018	2038.46	13580.24	1937	1947.80	5486.99	1692	1735.19	7828.20
reg5000.8	2025	2046.66	13324.79	1922	1947.70	5468.50	1709	1757.39	8308.68
reg5000.16	2026	2045.04	13260.26	1927	1946.27	5472.89	1691	1732.20	7685.04
cat.352	85	85.00	58.39	85	86.80	70.17	85	85.18	65.05
cat.702	57	60.32	128.62	83	87.18	197.15	55	59.00	141.88
cat.1052	83	87.76	277.63	137	142.98	319.83	76	90.47	219.26
cat.5252	162	167.37	5429.72	582	596.85	5304.24	161	193.21	3763.69
rcat.134	91	91.00	12.91	91	91.00	16.65	91	91.00	15.62
rcat.554	159	159.00	49.82	159	159.00	114.20	159	159.00	82.46
rcat.994	31	31.60	61.25	32	38.30	512.03	31	31.21	29.60
rcat.5114	475	481.52	1400.33	613	700.34	10626.14	483	488.09	3696.32
grid100.10	108	108.16	14.45	108	108.04	15.92	108	108.65	10.79
grid500.21	220	223.42	119.09	219	222.28	143.07	211	219.09	102.11
grid1000.20	316	319.98	325.43	318	319.76	406.37	316	318.97	244.60
grid5000.50	689	702.38	3389.93	684	699.54	3783.01	658	672.98	3345.94
w-grid100.20	128	128.00	14.18	128	128.00	16.34	128	128.26	18.37
w-grid500.42	262	267.70	125.71	263	266.68	137.42	256	263.47	118.52
w-grid1000.40	386	389.06	358.63	387	388.60	376.70	386	388.26	216.84
w-grid5000.100	837	849.58	3874.54	828	845.70	4274.58	811	825.27	3654.44

1. The best of 17,500 runs
2. Average of 50 runs, each of which is the best of 350 runs
3. CPU seconds on Pentium III 866MHz
4. From 100 runs

tems at Seoul National University.

References

[1] R. Battiti and A. Bertossi. Greedy, prohibition, and reative heuristics for graph partitioning. *IEEE Trans. on Computers*, 48(4):361–385, 1999.

[2] M. A. Breure. *Design Automation of Digital Systems.* Prentice-Hall, New York, 1972.

[3] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[4] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159, 1992.

[5] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.

[6] J. P. Cohoon, W. N. Martin, and D. S. Richards. A multi-population genetic algorithm for solving the *k*-part on hyper-cubes. In *Fourth International Conference on Genetic Algorithms*, pages 244–248, 1991.

[7] R. Collins and D. Jefferson. Selection in massively parallel genetic algorithms. In *Fourth International Conference on Genetic Algorithms*, pages 249–256, 1991.

[8] J. Cong and S.K. Lim. Multiway partitioning with pairwise movement. In *International Conference on Computer-Aided Design*, pages 512–516, 1998.

[9] M. Dell'Amico and F. Maffioli. A new tabu search approach to the 0-1 equicut problem. In *Meta-Heuristics 1995: The State of the Art*, pages 361–377. Kluwer Academic Publishers, 1996.

[10] C. Fiduccia and R. Mattheyses. A linear time heuristics for improving network partitions. In *19th IEEE/ACM Design Automation Conference*, pages 175–181, 1982.

[11] A. S. Fukunaga, J. H. Huang, and A. B. Kahng. On clustered kick moves for iterated-descent netlist partitioning. In *IEEE International Symposium on Circuits and Systems*, volume 4, pages 496–499, 1996.

[12] L. Hagen, J. H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(10):1199–1205, 1997.

[13] M. Harpal, M. Kishan, M. Chilukuri, and R. Sanjay. Genetic algorithms for graph partitioning and incremental graph partitioning. In *IEEE Proceedings of the Supercomputing*, pages 449–457, 1994.

[14] T.C. Hu and E. S. Kuh. *VLSI Curcuit Layout Theory and Design.* IEEE Press, New York, 1985.

[15] D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation. *Operations Research*, 37:865–892, 1989.

[16] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[17] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

[18] S. Kirkpatrick, Gelatt C. D. Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[19] G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Fourth International Conference on Genetic Algorithms*, pages 45–52, 1991.

[20] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.

[21] B. R. Moon and C. K. Kim. A two-dimensional embedding of graphs for genetic algorithms. In *International Conference on Genetic Algorithms*, pages 204–211, 1997.

[22] E. Rolland, H. Pirkul, and F. Glover. A tabu search for graph partitioning. *Annals of Operations Research*, 63, 1996.

[23] Y. Saab and V. Rao. Stochastic evolution: A fast effective heuristic for some genetic layout problems. In *27th IEEE/ACM Design Automation Conference*, pages 26–31, 1990.

[24] P. Sadayappan, F. Ercal, and J. Ramanujam. Partitionig graphs on message-passing machines by pairwise mincut. *Information Sciences*, pages 223–237, 1998.

[25] L. A. Sanchis. Multiple-way network partitioning. *IEEE Trans. on Computers*, 38(1):62–81, 1989.

[26] A. G. Steenbeek, E. Marchiori, and A. E. Eiben. Finding balanced graph bi-partitions using a hybrid genetic algorithm. In *IEEE Conference on Evolutionary Computation*, pages 90–95, 1998.

A Hybrid Genetic Algorithm for the MAX CUT Problem

Su-Hyang Kim
Information Technology Center
of LG-EDS Systems
236-1, Hyosung2-dong, Kyeyang-gu
Inchun, 407-042, Korea
`sukim@lgeds.lg.co.kr`

Yong-Hyuk Kim
School of Computer Science
& Engineering
Seoul National University
Shillim-dong, Kwanak-gu
Seoul, 151-742, Korea
`yhdfly@soar.snu.ac.kr`

Byung-Ro Moon
School of Computer Science
& Engineering
Seoul National University
Shillim-dong, Kwanak-gu
Seoul, 151-742, Korea
`moon@soar.snu.ac.kr`

Abstract

We propose a new hybrid genetic algorithm for MAX CUT graph partitioning. The algorithm includes a new local optimization heuristic with a new type of gain as the primary measure for vertex movement. We also present empirical comparisons of our algorithm with well known algorithms for graph partitioning. The suggested algorithm showed significant improvement over GW-SA, a state-of-the-art MAX CUT partitioning algorithm.

1 Introduction

It is of fundamental importance in combinatorial optimization to partition the vertices into two disjoint subsets of nearly equal size such that the sum of edge weights with two edge endpoints in different sets (cut size) is maximized or minimized. Given an undirected graph $G = (V, E)$ with edge weights $(w_{ij})_{(i,j)\in E}$, the maximum cut graph partitioning problem (MAX CUT) is that of finding a subset $S \subset V$ which maximizes the sum of edge weights in the cut $(S, V\backslash S)$.

Every graph has a finite number of cuts, so one can find the minimum or maximum weight cut in a graph by an exhaustive search that enumerates the sizes of all the cuts. This is not a practical approach for large graphs which arise in real-world applications since the number of cuts in a graph grows exponentially with the number of vertices. Although we can solve the min-cut problem without balance requirement in polynomial time using the maxflow-mincut algorithm [FF62], we have no such fortune when it comes to the MAX CUT problem. There is no known way to solve the problem optimally other than by exhaustive enumeration. The MAX CUT problem is one of Karp's original NP-complete problems [Kar72] and has been known to be NP-complete even if the problem is unweighted [GJS76], i.e., $w_{ij} = 1$ for every $(i, j) \in E$.

Since there is no algorithm that guarantees an optimal solution, a typical approach to solving such a problem is to find a ρ-approximation algorithm that delivers a solution at least ρ times the optimal value in polynomial time. Sahni and Gonzales [SG76] presented a $\frac{1}{2}$-approximation algorithm for the MAX CUT problem. Their greedy algorithm iterates through the vertices and decides which placement (S or $V\backslash S$) maximizes the cut of vertex v_i with respect to vertices v_1 to v_{i-1}. Since [SG76], a number of researchers have presented approximation algorithms for the MAX CUT problem [Vit81] [PT82] [HV91] [HL95], but no progress has been made. For more than twenty years a factor of 0.5 has been the best known polynomial time performance guarantee for the MAX CUT problem. A recent algorithm by Goemans and Williamson (GW) [GW95] guarantees a factor of 0.878 of the optimum. The significant improvements are due to the techniques of positive semidefinite programming and randomized rounding. However, solving semidefinite programming is computationally expensive. Homer and Peinado [HP97] gave a parallelized version of GW.

The MAX CUT problem has applications in various fields. It has been observed that one of the phases (the layer assignment problem) in the design process for VLSI chips and printed circuit boards can be reduced to the MAX CUT problem [BGJR88] [Pin84]. One of the most famous applications of the problem comes from a classical application to statistical physics [BGJR88]. It is concerned with the exact determination of a minimal energy configuration of a spin glass under no exterior field and under a continuously varying exterior magnetic field. For a comprehensive survey of the MAX CUT problem, refer to [PT93].

In this paper, we suggest a new hybrid genetic algorithm for the MAX CUT graph partitioning. We focus on the local improvement for GA. As a local optimization engine for the hybrid GA, we use a variation of the traditional framework of the Fiduccia-Mattheyses algorithm [FM82]. We compare the hybrid GA against some algorithms in the literature: the Goemans-Williamson algorithm, the Fiduccia-Mattheyses algorithm, and some variants of the FM algorithm.

In Section 2, we summarize the Goemans-Williamson algorithm and the Fiduccia-Mattheyses algorithm. In Section 3, we describe new local improvement algorithms for the MAX CUT problem. A hybrid genetic algorithm is described in Section 4. In Section 5, we present our experimental results. Finally, we present our conclusions in Section 6.

2 Previous Works

2.1 Goemans-Williamson Algorithm (GW)

A randomized version of the greedy algorithm of Sahni and Gonzalez [SG76] can be stated as follows. Given a graph, generate cuts according to the uniform distribution. By linearity of expectation, the expected cut size under this distribution is half the sum of all edge weights, and thus at least half the size of the optimum cut.

Goemans and Williamson [GW95] improved this factor from 0.5 to 0.878 by generating cuts from a more sophisticated distribution. Each vertex is represented by a $|V|$-dimensional unit vector v_i. The first step of their algorithm (called GW) is to find a vector configuration, i.e., a set of $|V|$ vectors in $\mathcal{R}^{|V|}$ which solves

$$\max Z_v = \frac{1}{2}\sum_{i<j} w_{ij}(1 - \langle v_i, v_j\rangle)$$
$$\text{subject to } ||v_i||_2 = 1 \quad \forall v_i \in V \qquad (1)$$

where $\langle\cdot,\cdot\rangle$ is the inner product. A near-optimal solution for (1) can be found in polynomial time using a semidefinite programming algorithm and incomplete Cholesky decomposition.

The second step of the GW algorithm is to uniformly generate random hyperplanes through the origin. A hyperplane, given by its normal vector n, separates the vectors into the two disjoint sets $L = \{v_i : \langle v_i, n\rangle \geq 0\}$ and $R = \{v_i : \langle v_i, n\rangle < 0\}$. This defines a cut. The analysis of Goemans and Williamson shows that the expected cut size under this distribution is at least 0.878 times the size of the optimum cut.

2.2 Fiduccia-Mattheyses algorithm (FM)

The de facto benchmark graph partitioning algorithm for almost 25 years has been the famous local search heuristic by Kernighan and Lin (KL). The KL algorithm [KL70] proceeds in a series of passes. During each pass, the algorithm improves on an initial solution by swapping pairs of vertices to create new solutions. Since there is a restriction in subset size, KL maintain balance by swapping equal-sized subsets of vertices. Fiduccia and Mattheyses [FM82] provided a KL-inspired algorithm which allows unbalanced partitions to some degree and reduces the time per pass. The main difference between KL and FM lies in the fact that a new partition in FM is derived by moving a single vertex, instead of KL's pair swap, and in FM's efficient data structure.

Let (A, B) be an initial solution of $G = (V, E)$. We define the gain of a vertex v to be the cut-size reduction by moving v to the opposite set. FM selects one vertex that has the highest gain and moves it to the opposite side. At the beginning of a pass, all vertices are unlocked. After moving a vertex, the vertex is locked for the rest of the pass. FM iteratively moves unlocked vertices until all vertices are locked. The best partition during the pass is returned as a new solution. Another pass starts with this new solution. The algorithm terminates when one or a few passes fail to find a better solution. Due to efficient gain bucket management, a pass of FM can be completed in $O(|E|)$ time.

3 Proposed Algorithms

3.1 A Variation of FM (MAX-FM)

We use a variation of the FM implementation and its gain bucket data structure. It may be recalled that Kernighan and Lin developed a heuristic procedure for mincut partitioning of a graph into two equal-sized subsets. In contrast, our goal is to find a maxcut with no restriction on the size of the resulting partition. We start with a randomly generated initial partition. Then we try to improve the partition by moving a vertex. Since there is no restriction on the partition size, it is not necessary to swap vertices or to alternately select a vertex in each partition. In practice, the size of each partition is nearly balanced even without any effort to adjust the balance.

Let (A, B) be an initial partition of $G = (V, E)$. Define the gain or general gain g_v of a vertex v to be the cut-size increase by moving v to the opposite set. Formally,

$$g_v = \mathit{indegree}(v) - \mathit{outdegree}(v)$$

where $\mathit{indegree}(v)$ is the weight sum of edges incident to vertices in the same set, and $\mathit{outdegree}(v)$ is the weight sum of edges incident to vertices in the opposite set. When a series of passes are terminated, the partition which has the maximum cut size is returned.

```
do {
    Compute $g_v$ for every vertex $v \in V$;
    Make two gain lists for each $a \in S$ and $b \in V \setminus S$;
    $Q = \phi$;
    for $i = 1$ to $|V| - 1$ {
        Choose $v_i \in V - Q$ such that $g_{v_i}$ is maximal;
        $Q = Q \cup \{v_i\}$;
        if $v_i \in S$ then
            for each $v \in S - Q$ adjacent to $v_i$
                $g_v = g_v - 2\, w_{v_i v}$;
            for each $v \in V \setminus S - Q$ adjacent to $v_i$
                $g_v = g_v + 2\, w_{v_i v}$;
        else if $v_i \in V \setminus S$ then
            for each $v \in S - Q$ adjacent to $v_i$
                $g_v = g_v + 2\, w_{v_i v}$;
            for each $v \in V \setminus S - Q$ adjacent to $v_i$
                $g_v = g_v - 2\, w_{v_i v}$;
    }
    Choose $k \in \{1, 2, \ldots, |V| - 1\}$ to maximize $\sum_{i=1}^{k} g_{v_i}$;
    Move all the vertices in the subset $\{v_1, v_2, \ldots, v_k\}$
        to their opposite side;
} until (there is no improvement)
```

Figure 1: The FM algorithm for MAX CUT

The structure of the variation of the FM algorithm is given in Figure 1.

3.2 MAX Lock-Gain Based Algorithm (MAX-LG)

We proposed the Lock-Gain based algorithm (LG) [KM00]. LG showed a dramatic performance improvement for mincut partitioning on sparse geometric graphs and caterpillar graphs which have some unit clusters. We define the lock gain l_v of the vertex v to be the gain by moving the vertex only with respect to the locked vertices.

The lock gain considers the fixed part of the gain since locked vertices do not move any more. That is, the general gain g_v consists of two types of gain: fixed gain and changeable gain. The lock gain based heuristic gives preference to unchangeable gain over changeable gain. For the MAX CUT partitioning, we redefine the lock gain l_v to be the cut-size "increase" instead of cut-size reduction.

3.3 MAX Ratio-Gain Based Heuristic (MAX-RG)

MAX-RG uses the sum of the lock gain and general gain in an appropriate ratio as the measure for selecting a vertex to move. We define the ratio gain r_v of vertex v to be the sum of the general gain and lock gain with some ratio, i.e.,

$$r_v = g_v + \alpha \times l_v$$

where $\alpha \in \{0, 1, \ldots, D_{max}\}$ and D_{max} is the maximum vertex degree in the graph. Since MAX-LG works well for sparse graphs, we use a large ratio for the sparse graph. The density of a graph is defined by the average vertex degree:

$$\delta(G) = \frac{\sum_{i=1}^{|V|} deg(v_i)}{|V|}$$

where $deg(v_i)$ is the degree of vertex $v_i \in V$. MAX-RG has the same framework as MAX-FM except for the measure used for vertex selection.

It starts with a random initial solution and improves on it through a number of passes. At the beginning of a pass, every vertex is unlocked and the lock gain is set to zero. The ratio gain is equal to the general gain. We choose the vertex v which has the maximum ratio gain. Once a vertex v is chosen, it is assumed to move to the opposite side and becomes locked. After a vertex is locked, the lock gains and general gains of its adjacent vertices are adjusted. The moving process is iterated until a sequence of vertices $V = (v_1, v_2, \ldots, v_{|V|})$ is decided. The algorithm then finds a subset of V, $X = \{v_1, v_2, \ldots, v_k\}$ such that $\sum_{i=1}^{k} g_{v_i}$ is maximized, and moves the subset X. Note that although the vertices $v_1, v_2, \ldots, v_{|V|}$ are determined on the basis of ratio gains, the final set X is chosen by general gains. The acquired solution becomes an initial solution for the next pass. It continues until there is no improvement.

3.4 Tuning of Ratio Factor

For the local search heuristic algorithm, we experimented with many graphs having various ratios of lock gain and general gain. If $\alpha = 0$, r_v is equal to the general gain g_v and the algorithm works exactly like MAX-FM. When $\alpha = 1$, the algorithm considers general gain and lock gain equally. The main difference from MAX-LG is that MAX-RG considers general gain as well as lock gain, simultaneously. When α is bigger, the algorithm worked well on the sparse graphs, but produced some degradation on the dense graphs. If $\alpha = D_{max}$, then r_v is bound to the lock gain and the algorithm works exactly like MAX-LG. The experimental observations above are consistent with [KM00], where LG worked well on sparse graphs with some unit clusters.

In the first experiment, we fixed the ratio factor α to be in inverse proportion to the density of a graph G as follows:

$$\alpha = \begin{cases} \lfloor \beta/\gamma \rfloor - \lfloor \delta(G)/\gamma \rfloor, & \text{if } \beta > \delta(G) \\ 0 \qquad\qquad\qquad\quad , & \text{otherwise.} \end{cases}$$

In our experiment, we set β to 40 and γ to 10. In the hybrid GA, instead of adjusting the ratio factor α manually, we set α to be a part of the chromosome for GA. Initially, the ratio factor α is randomly generated in the range of the above. As the population converges, a reasonable factor for the solution will survive. We select the α that has the largest fitness sum in the population.

3.5 Implementation

We use the bucket data structure to maintain two separate ratio gain lists. This data structure allows constant time selection of a vertex and fast gain update after each move. Since every general gain and lock gain is an integer in the range $[-D_{max}, D_{max}]$, every ratio gain is also an integer in the range $[-(\alpha+1) \times D_{max}, (\alpha+1) \times D_{max}]$. This makes possible the efficient bucket management of ratio gains. At the beginning of a pass, all vertices are inserted into the bucket of their general gain value. For each step in the pass, a vertex is selected in a bucket and deleted from the bucket. After the vertex moves to the opposite side, the gains (general gain, lock gain, and ratio gain) of unlocked vertices adjacent to the moved vertex are updated. The update of a vertex is carried out by removing it from its ratio gain bucket and inserting it into the bucket of its new ratio gain value. If one of these inserted vertices has a new ratio gain that is larger than the current max ratio gain, then the pointer to the max ratio gain (denoted by MaxRatioGain) is replaced by this new value. If the bucket list with MaxRatioGain becomes empty, then the MaxRatioGain is decreased until it indexes a non-empty bucket.

4 A Hybrid Genetic Algorithm

The general structure of hybrid steady-state genetic algorithms is used. The population size is set to fifty and other parameters are described in the following.

- *Encoding*: A chromosome corresponds to a partition $(S, V \backslash S)$ of the graph $G = (V, E)$. The number of genes in a chromosome is equal to $|V| + 1$. Each gene corresponds to a vertex in the graphs. A gene has value zero if the vertex is on the set S, otherwise its value is one. The $(|V|+1)^{th}$ gene corresponds to a ratio factor α. For other GAs that do not tune the ratio factor α, the number of genes is $|V|$.

- *Initialization*: The algorithm first creates fifty initial solutions at random.

- *Selection*: We assign to each chromosome in the population a fitness value calculated from its cut size. We use a proportional selection scheme.

- *Genetic operators*: A crossover operator creates a new offspring by combining parts of the two parents. In our experiments, we use crossover with five cut points and no mutation.

- *Local optimization*: After crossover, we apply a local optimization on the offspring. We use MAX-FM, MAX-LG, and MAX-RG as the local optimization.

- *Replacement*: After generating an offspring and applying a local optimization, the GA replaces a member of the population with the offspring. We first try to replace one of the parents [Cav70]; the offspring tries to first replace the more similar parent, measured in bitwise difference and, if it fails, then it tries to replace the other parent. Replacement is done when the offspring is superior to one of its parents. If the offspring is worse than both parents, we replace the worst member of the population (Genitor-style replacement [WG88]). This combination of preselection [Cav70] and Genitor-style replacement [WG88] was used in [BM94] and showed good performance.

- *Stopping condition*: We use the number of consecutive fails to replace one of the parents as a stopping condition [BM94]. We set this number to be twenty in our algorithm.

5 Experimental Results

5.1 Test Beds and Test Environment

In this section, we describe our experiments and present the results for various graphs including negative edge weights. The types of graphs on which we experimented are graphs used by Johnson *et al.* [JAMS89], sparse random graphs used by Homer and Peinado [HP97], and graphs derived from circuit design problems.

- G$n.d$: A random graph on n vertices, where an edge is placed between any two vertices with probability p independent of all other edges. The probability p is chosen so that the expected vertex degree, $p(n-1)$, is d.

- U$n.d$: A random geometric graph on n vertices that lie in the unit square and whose coordinates are chosen uniformly from the unit interval. There

Table 1: Comparison of GW and MAX-FM

	GW		MAX-FM			GW-SA			MAX-FM-GA		
Graphs	Best	CPU	Best	Average	CPU	Best	Average	CPU	Best	Average	CPU
G500.2.5	**564**	1806	571	557.70	0.00	568	566.6	2541.4	574	**572.92**	2.20
G500.05	980	1839	1003	**984.01**	0.00	995	987.4	2598.6	1008	**1006.44**	3.24
G500.10	1687	1582	1733	**1707.97**	0.01	1706	1695.0	2354.4	1735	**1733.25**	4.84
G500.20	3291	1666	3386	**3347.10**	0.01	3312	3303.0	2454.0	3390	**3387.60**	9.33
G1000.2.5	**1150**	10488	1162	1138.58	0.01	1164	1155.0	13489.4	1173	**1167.58**	6.31
G1000.05	1982	7048	2034	**2001.70**	0.01	1996	1990.8	10025.6	2053	**2047.68**	9.94
G1000.10	3578	9415	3688	**3637.92**	0.02	3603	3591.6	12452.0	3703	**3695.95**	18.23
G1000.20	6537	15778	6711	**6635.45**	0.05	6575	6557.0	19430.4	6729	**6722.36**	25.58
U500.05	885	1503	897	**889.41**	0.00	898	895.4	2290.2	900	**899.22**	2.43
U500.10	1520	2396	1540	**1527.73**	0.01	1537	1531.2	3200.0	1546	**1543.39**	4.49
U500.20	2731	2641	2778	**2759.65**	0.01	2752	2742.4	3499.2	2783	**2780.92**	7.68
U500.40	5107	4879	5180	**5151.28**	0.03	5135	5125.8	5893.6	5181	**5180.58**	15.09
U1000.05	1673	7970	1705	**1685.24**	0.01	1702	1699.0	11232.2	1709	**1704.21**	6.63
U1000.10	3005	12871	3052	**3028.99**	0.01	3037	3027.2	16166.6	3071	**3063.73**	12.99
U1000.20	5623	21857	5723	**5686.93**	0.04	5661	5652.8	27582.6	5736	**5730.41**	31.41
U1000.40	10391	67936	10556	**10492.62**	0.09	10444	10414.4	73497.6	10560	**10556.11**	68.01
R1000	3558	11503	3675	**3618.49**	0.02	3605	3584.8	13820.2	3687	**3679.91**	17.75
R2000	7030	104328	7235	**7157.81**	0.07	7063	7051.0	111198.8	7308	**7289.49**	77.44
R3000	8356	617542	10863	**10765.34**	0.13	8488	8419.2	680640.8	10987	**10950.28**	163.30
R4000	11092	134669	14460	**14341.70**	0.19	11158	11132.4	157872.0	14626	**14584.90**	264.69
R5000	13351	290451	17988	**17836.34**	0.26	13440	13402.2	320128.8	18202	**18146.35**	392.11
R6000	15580	96836	21624	**21474.62**	0.34	15755	15673.2	144868.6	21937	**21864.62**	557.89
R8000	19639	73321	28684	**28518.51**	0.50	20399	20168.8	152606.4	29140	**29030.74**	874.96
via.c1n	**5868**	7659	6132	5700.57	0.01	6150	6015.6	18369.0	6150	**6148.38**	5.96
via.c2n	**6954**	11855	7074	6585.28	0.01	7026	6982.8	28869.2	7098	**7087.56**	7.55
via.c3n	**6640**	27182	6802	6029.28	0.01	6856	6737.2	56270.2	6898	**6818.32**	12.01
via.c4n	**9954**	27268	10020	9120.26	0.02	10062	10038.0	64484.2	10098	**10078.20**	11.02
via.c5n	**7686**	18507	7848	7070.39	0.01	7854	7810.8	45189.4	7956	**7883.52**	11.54
via.c1y	7284	4379	7746	**7622.55**	0.01	7728	7546.8	13714.6	7746	**7746.00**	2.23
via.c2y	7728	5991	8226	**8089.36**	0.01	8226	8120.4	19785.8	8226	**8226.00**	2.66
via.c3y	8055	4476	9502	**9347.55**	0.03	9453	9105.6	26692.0	9502	**9502.00**	4.88
via.c4y	10890	4661	12516	**12377.07**	0.03	12516	11882.4	28318.8	12516	**12516.00**	4.03
via.c5y	8940	3578	10248	**10018.18**	0.03	10236	9864.0	21534.2	10248	**10248.00**	5.50

The GW results were acquired from 1 run.

The MAX-FM results were acquired from 1,000 runs.

The GW-SA results were acquired from 5 runs, with 1,000 steps per run.

The MAX-FM-GA results were acquired from 100 runs.

is an edge between two vertices if their Euclidean distance is t or less, where $d = n\pi t^2$ is the expected vertex degree.

- Rn : A sparse random graph in which the edge probability is set to $p = 10/n$, where $n = |V|$. This was used in [HP97], corresponding to random graph class C in Goemans and Williamson [GW95].
- via.c : A graph derived from VLSI problems. It includes negative edge weights. This was used in [HP97].

All programs were run on Pentium III 500MHz computers with Linux operating systems.

5.2 Performance

Table 1 shows the performance of GW and MAX-FM. GW-SA is a simulated annealing algorithm which uses GW's results as initial solutions. MAX-FM-GA is a genetic algorithm which uses MAX-FM as the local optimization engine. As a local search heuristic, MAX-FM outperformed GW for most graphs. Due to the burden of matrix computation, GW took at least order of 10^5 times more than MAX-FM. In [HP97], GW was improved by combining with simulated annealing [KGV83] for some graphs, which is consistent in this experiment. MAX-FM-GA clearly outperformed the others.

Table 2 compares the performance of MAX-FM, MAX-LG, and MAX-RG. MAX-RG overall performed best among them. For sparse random graphs of [HP97], MAX-LG performed best.

Table 3 gives a comparison of hybrid GAs. MAX-LG-GA and MAX-RG-GA performed comparably well. MAX-FM was not comparable with MAX-LG and MAX-RG as a local optimization heuristic as well as in the context of hybrid GAs. Table 3 also contains the generations which equals the number of local-optimization calls. It is notable that the hybrid GAs are much faster than a single run of GW (see Table 1).

Table 2: Comparison of MAX-FM, MAX-LG, and MAX-RG over 1,000 Runs

	MAX-FM			MAX-LG			MAX-RG		
Graphs	Best	Average	CPU	Best	Average	CPU	Best	Average	CPU
G500.2.5	571	557.70	0.00	574	566.30	0.01	574	**567.44**	0.01
G500.05	1003	984.01	0.00	1004	989.62	0.01	1007	**993.72**	0.01
G500.10	1733	1707.97	0.01	1732	1710.29	0.02	1732	**1713.40**	0.02
G500.20	3386	3347.10	0.01	3386	3349.28	0.03	3385	**3350.94**	0.04
G1000.2.5	1162	1138.58	0.01	1168	1155.83	0.04	1171	**1158.19**	0.02
G1000.05	2034	2001.70	0.01	2042	2018.93	0.05	2043	**2022.39**	0.03
G1000.10	3688	3637.92	0.02	3691	3651.05	0.07	3695	**3652.87**	0.07
G1000.20	6711	6635.45	0.05	6702	**6642.57**	0.13	6705	6641.90	0.15
U500.05	897	889.41	0.00	900	896.48	0.02	900	**897.13**	0.01
U500.10	1540	1527.73	0.01	1543	1534.86	0.02	1546	**1536.39**	0.02
U500.20	2778	2759.65	0.01	2780	2765.04	0.03	2781	**2768.75**	0.04
U500.40	5180	5151.28	0.03	5180	5156.03	0.06	5180	**5161.13**	0.06
U1000.05	1705	1685.24	0.01	1710	1704.57	0.07	1711	**1705.24**	0.03
U1000.10	3052	3028.99	0.01	3070	3052.01	0.07	3068	**3054.22**	0.06
U1000.20	5723	5686.93	0.04	5731	5703.59	0.11	5732	**5711.36**	0.15
U1000.40	10556	10492.62	0.09	10553	10506.85	0.22	10556	**10512.46**	0.22
R1000	3675	3618.49	0.02	3676	3634.23	0.06	3675	**3635.25**	0.07
R2000	7235	7157.81	0.07	7270	**7205.67**	0.27	7269	7199.88	0.25
R3000	10863	10765.34	0.13	10930	**10847.63**	0.60	10908	10838.24	0.50
R4000	14460	14341.70	0.19	14547	**14463.10**	1.04	14534	14446.82	0.80
R5000	17988	17836.34	0.26	18010	**18000.70**	1.64	18067	17974.96	1.11
R6000	21624	21474.62	0.34	21787	**21685.87**	2.40	21762	21651.97	1.50
R7000	25395	25212.08	0.42	25588	**25470.04**	3.19	25554	25403.97	2.02
R8000	28684	28518.51	0.50	28940	**28814.26**	4.56	28897	28768.24	2.40
via.c1n	6132	5700.57	0.01	6150	6133.76	0.02	6150	**6146.15**	0.02
via.c2n	7074	6585.28	0.01	7098	7081.46	0.02	7098	**7094.03**	0.03
via.c3n	6802	6029.28	0.01	6898	6874.79	0.04	6898	**6877.08**	0.04
via.c4n	10020	9120.26	0.02	10098	10091.86	0.03	10098	**10096.96**	0.04
via.c5n	7848	7070.39	0.01	7956	7925.02	0.03	7956	**7941.74**	0.04
via.c1y	7746	7622.55	0.01	7746	7743.59	0.02	7746	**7743.69**	0.05
via.c2y	8226	8089.36	0.01	8226	**8224.98**	0.02	8226	8224.79	0.06
via.c3y	9502	9347.55	0.03	9502	9496.49	0.04	9502	**9498.42**	0.09
via.c4y	12516	12377.07	0.03	12516	**12515.66**	0.04	12516	12514.77	0.08
via.c5y	10248	10018.18	0.03	10248	**10239.16**	0.04	10248	10234.62	0.10

6 Conclusions

We have studied methods of improving the performance of algorithms to solve the maximum cut graph partitioning problem.

First, we found an improvement by using the ratio gain as the primary measure for vertex movement. By adjusting the ratio factor, we could obtain better results. Second, by combining with the framework of genetic algorithms, we achieved further improvement. In GA, we attempted to adaptively determine proper ratio factors in an appropriate time. Finding a good strategy for adjusting the ratio will be a useful subject for further study.

Our local optimization heuristic algorithm can be combined with other meta-heuristics such as SA [KGV83], tabu search [Glo89], LSMC [MOF91], etc. We believe that our algorithm will likely lead to improvement on existing results in those frameworks, as it has done in the framework of GA. Experimentation with respect to these methods is left for future study.

Acknowledgments

We would like to thank M. Peinado for providing us with benchmark graphs and the source code of GW-SA.

This research was supported in part by KOSEF through Statistical Research Center for Complex Systems at Seoul National University.

References

[BGJR88] F. Barahona, M. Grotschel, M. Junger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operational Research*, 36:493–513, 1988.

[BM94] T. N. Bui and B. R. Moon. A genetic algorithm for a special class of the quadratic assignment problem. *The Quadratic Assignment and Related Problems DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 16:99–116, 1994.

[Cav70] D. Cavicchio. *Adaptive Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor, MI, 1970. Unpublished.

[FF62] L. R. Jr. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

Table 3: Comparison of MAX-FM-GA, MAX-LG-GA, and MAX-RG-GA

	MAX-FM-GA			MAX-LG-GA			MAX-RG-GA		
Graphs	Best	Average	Gen	Best	Average	Gen	Best	Average	Gen
G500.2.5	574	572.92	568	574	**573.98**	340	574	573.96	365
G500.05	1008	1006.44	590	1008	1007.23	392	1008	**1007.68**	451
G500.10	1735	1733.25	539	1735	1733.42	395	1735	**1734.05**	452
G500.20	3390	3387.60	561	3390	3388.81	533	3390	**3389.28**	641
G1000.2.5	1173	1167.58	736	1173	1170.37	381	1173	**1170.90**	424
G1000.05	2053	2047.68	852	2053	2048.23	461	2054	**2048.97**	536
G1000.10	3703	3695.95	826	3703	3697.20	552	3702	**3697.43**	643
G1000.20	6729	6722.36	909	6730	6722.94	654	6729	**6723.74**	772
U500.05	900	899.22	427	900	**900.00**	250	900	**900.00**	295
U500.10	1546	1543.39	511	1546	**1545.13**	337	1546	1545.09	350
U500.20	2783	2780.92	553	2783	2781.44	377	2783	**2782.33**	498
U500.40	5181	5180.58	485	5181	5180.24	399	5181	**5180.68**	484
U1000.05	1709	1704.21	538	1711	1710.85	285	1711	**1710.86**	258
U1000.10	3071	3063.73	667	3074	**3071.05**	434	3073	3070.87	431
U1000.20	5736	5730.41	711	5737	5733.41	437	5737	**5734.09**	490
U1000.40	10560	10556.11	741	10560	10556.97	490	10560	**10557.58**	608
R1000	3687	3679.91	831	3685	3682.42	581	3687	**3682.72**	634
R2000	7308	7289.49	1270	7307	**7300.01**	821	7308	7299.25	933
R3000	10987	10950.28	1435	10991	**10972.55**	733	10989	10962.63	871
R4000	14626	14584.90	1594	14649	**14608.74**	742	14628	14590.93	683
R5000	18202	18146.35	1801	18212	**18186.21**	790	18203	18160.22	825
R6000	21937	21864.62	2100	21953	**21921.14**	921	21957	21897.99	1004
R7000	25735	25670.34	2257	25771	**25730.46**	881	25757	25694.30	780
R8000	29140	29030.74	2353	29142	**29102.38**	892	29143	29053.01	730
via.c1n	6150	6148.38	710	6150	**6150.00**	150	6150	**6150.00**	198
via.c2n	7098	7087.56	725	7098	**7098.00**	143	7098	**7098.00**	196
via.c3n	6898	6818.32	792	6898	**6898.00**	200	6898	**6898.00**	275
via.c4n	10098	10078.20	658	10098	**10098.00**	111	10098	**10098.00**	178
via.c5n	7956	7883.52	830	7956	**7956.00**	251	7956	**7956.00**	272
via.c1y	7746	**7746.00**	216	7746	**7746.00**	105	7746	**7746.00**	140
via.c2y	8226	**8226.00**	197	8226	**8226.00**	100	8226	**8226.00**	133
via.c3y	9502	**9502.00**	220	9502	**9502.00**	158	9502	**9502.00**	170
via.c4y	12516	**12516.00**	166	12516	**12516.00**	100	12516	**12516.00**	122
via.c5y	10248	**10248.00**	292	10248	**10248.00**	138	10248	**10248.00**	142

Data from 100 runs.

"Gen" means the average number of generations.

[FM82] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comp. Sci.*, 1:237–267, 1976.

[Glo89] F. Glover. Tabu search - part i. *ORSA journal on Computing*, 1:190–206, 1989.

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):1115–1145, 1995.

[HL95] T. Hofmeister and H. Lefmann. A combinatorial design approach to maxcut. *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science*, 1995.

[HP97] S. Homer and M. Peinado. Design and performance of parallel and distributed approximation algorithms for maxcut. *Journal of Parallel and Distributed Computing*, 46:48–61, 1997.

[HV91] D. J. Haglin and S. M. Venkatesan. Approximation and intractability results for the maximum cut problem and its variants. *IEEE Trans. on Computers*, 40:110–113, 1991.

[JAMS89] D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part 1, graph partitioning. *Operations Research*, 37:865–892, 1989.

[Kar72] R. M. Karp. *Reducibility among combinatorial problems*, pages 85–103. Plenum Press, New York, 1972.

[KGV83] S. Kirkpatrick, C. D. Jr. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KL70] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell systems Technical Journal*, 49:291–307, 1970.

[KM00] Y. H. Kim and B. R. Moon. A hybrid genetic search for graph partitioning based on lock gain. In *Genetic and Evolutionary Computation Conference*, pages 167–174, 2000.

[MOF91] O. C. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

[Pin84] R. Y. Pinter. Optimal layer assignment for interconnect. *Journal of VLSI Computing Systems*, 1:123–137, 1984.

[PT82] S. Poljak and Z. Tuza. A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem. *Canadian Journal of Mathematics*, 34:519–524, 1982.

[PT93] S. Poljak and Z. Tuza. *Maximum cuts and largest bipartite subgraphs*, volume 20. American Mathematical Society, 1993.

[SG76] S. Sahni and T. Gonzalez. P-complete approximation problem. *Journal of the Association for Computing Machinery*, 46:48–61, 1976.

[Vit81] P. M. Vitanyi. How well can a graph be n-colored? *Discrete Mathematics*, 34:69–80, 1981.

[WG88] D. Whitley and J. Kauth. Genitor. A different genetic algorithm. In *In Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, 1988.

Benchmark Problem Generators and Results for the Multiobjective Degree-Constrained Minimum Spanning Tree Problem

Joshua D. Knowles and David W. Corne
Department of Computer Science, University of Reading, UK
J.D.Knowles@reading.ac.uk, D.W.Corne@reading.ac.uk
FAX: +44(0) 118 975 1994, TEL: +44 (0) 118 931 8983
http://www.rdg.ac.uk/~ssr97jdk

Abstract

Finding a minimum-weight spanning tree (MST) in a graph is a classic problem in operational research (OR) with important applications in network design. In this paper, we consider the degree-constrained multiobjective MST problem, which is NP-hard. We present several different parameterized problem generators for producing MST instances with different problem features, including any number of objectives, varying degrees of convexity and non-convexity in the Pareto front, and edge weight combinations that mislead greedy approaches. As well as being useful for the OR community, these generators are well-suited to provide problems to form part of a wider (evolutionary) multiobjective test problem suite, where constrained and NP-hard combinatorial problems are sometimes poorly represented. Fifteen instances are generated using the presented methods, and benchmark results on these instances for a multiobjective EA, AESSEA, are presented. These are compared with results from two different non-EA methods. All of our problem instances, generators, and solution sets will be made available for use by other researchers.

1 Introduction

For many years, minimum spanning tree (MST) problems have been of great interest to the operational research community. More recently, the multiobjective minimum spanning tree (mc-MST)[1] problem, in which there are multiple weights defined on each edge, and which is NP-hard, has become subject of increasing interest. Several papers on this subject [3, 5, 14] have proposed approximate polynomial algorithms, and exact methods, for tackling the problem. With growing interest in the evolutionary algorithm (EA) community in multiobjective optimization, it seems likely that this application should now become the focus of more EA approaches also. The first genetic algorithm (GA) for the mc-MST (in which a Prüfer number encoding is used) was proposed by Zhou and Gen [15]. The problems used to test their algorithm were, however, very simple and could be much better solved using good exact methods [14] (smaller problems), or heuristic approximation methods [3, 5] (larger problems). Nonetheless, some variants of the mc-MST may not be easily or efficiently solved by exact methods. One example is when the number of objectives is greater than two. It is then relatively straightforward for a multiobjective EA to be applied, whereas some exact methods [14] and heuristics have only been developed for the bi-objective case. Furthermore, when other constraints need to be incorporated then this may be achieved relatively easily in an evolutionary algorithm but not in some of the pure heuristic approaches. Unfortunately, these more difficult problems are not generally available and have not yet been considered in algorithm studies.

In order to aid in the further development of good evolutionary algorithms and other metaheuristics for a full range of variants of the mc-MST, it would be useful to have a collection of parameterized problem generators to provide benchmark problems that could be used in comparative studies. In this paper we present such a collection of simple problem generators that can provide problems with correlated and anti-correlated weights (which critically affect the shape of the Pareto front, and therefore the applicability of different methods), problems with large regions that have no solutions on the convex hull of the Pareto front, and problems which are difficult to solve when a degree constraint is additionally imposed. All of the gener-

---

[1] It is often called the multi-criterion MST, hence mc-MST.

ators can generate problems of different sizes, with a differing degree of sparsity, and with any number of objectives (except for the concave graph generator).

The generators proposed may also be used for testing the general strengths and weaknesses of multiobjective EAs, as part of a wider test problem suite. Much progress has been made recently in improving the variety, difficulty, and actual use of, test problem suites by researchers in the evolutionary multiobjective optimization (EMO) community. However, there is still a shortage of parameterized combinatorial problems, constrained problems, and problems with large numbers of objectives. The generators proposed here can provide problems with all these features, separately as well as together.

In this paper, the different generators are used to provide an initial set of 15 different problem instances, each with just two objectives. These instances are then tackled using a multiobjective EA called AESSEA, which is described in detail in a recent paper by us [9], where it was used to solve some much simpler random-weight mc-MST problems. The AESSEA results are also compared with an enumerative algorithm (smaller problems) and a polynomial-time iterated heuristic approach (larger problems) to provide the first benchmark results over a wide range of mc-MST problem types with constraints. All of the problem instances, generators, and results sets are available for others to use by e-mail contact with the first author.

The remainder of the paper is organized as follows: Section 2 defines the mc-MST problem and a degree-constrained variant. In Section 3, the different problem generators are described. Section 4 then describes AESSEA, the encoding and operators used, and the other non-EA algorithms. Section 5 provides details of the experimental method including all parameter settings, and describes the statistical method used to analyze the results. Section 6 presents the results, and Section 7 concludes.

2 Multiobjective Degree-Constrained Minimum Spanning Tree Problem

A spanning tree of an undirected, connected graph, $G = (V, E)$, is a subgraph $T = (V, E_T), E_T \subset E$ that contains all vertices in V and connects them with exactly $|V| - 1$ edges, so that there are no cycles. If G is complete, then the set S of spanning trees T of G has $|S| = |V|^{|V|-2}$ members. If each edge $(i,j) \in E$ has $K > 1$ associated non-negative real numbers, representing K attributes defined on it and denoted with $\mathbf{w}_{i,j} = (w^1_{i,j}, w^2_{i,j}, \ldots, w^K_{i,j})$, then the mc-MST problem may be defined as:

$$\begin{aligned} \text{“minimize” } \mathbf{W} &= (W^1, W^2, \ldots, W^K) \\ \text{with } W^k &= \sum_{(i,j) \in E_T} w^k_{i,j}, \; k \in 1..K \end{aligned} \qquad (1)$$

where the term ‘minimize’ is in quotation marks to indicate that it may not be possible to find a single solution that is minimal on all the components of $\mathbf{W}$. Instead, one is required to find a set of spanning trees $S^* \subset S$, called the Pareto optimal set, with the property that:

$$\forall T^* \in S^* \bullet \; \nexists T \in S \bullet T \prec T^* \qquad (2)$$

where $T \prec T^* \iff \forall k \in 1..K \bullet W^k \leq W^{k^*} \wedge \exists\, k \in 1..K \bullet W^k < W^{k^*}$. The expression $T \prec T^*$ is read as T *dominates* T^*, and solutions in the Pareto optimal set are also known as efficient or admissible solutions.

If there is, in addition, a constraint d on the maximum vertex degree in the spanning tree, then the problem is called the multiobjective degree-constrained minimum spanning tree (mc$d$-MST) problem.

3 Problem generators

We propose generators for the following types of non-Euclidean problems:

Random Random (uncorrelated) integer or real number weighted graphs;

Correlated: Random correlated real number weighted graphs;

Anti-Correlated: Random anti-correlated real number weighted graphs;

M-Correlated: Correlated real number weighted graphs with high vertex degree in the underlying MST;

Concave: Real number weighted graphs that have a large concave region in their Pareto front.

All of the above can be generated as either sparse graphs, or complete graphs. We consider only complete graphs in this paper. Similarly, all but the concave graphs can be generated with an arbitrary number of objectives, K, although here we restrict our attention to bi-objective problems only.

Random

The random graph generator simply sets each component of each edge weight vector to a value drawn from a uniformly random distribution within some range, $\mathcal{U}(min, max)$. In this paper we do not consider graphs of this type but we have already shown [9] that our AESSEA algorithm using a direct coding and specialized operators is superior to the AESSEA algorithm using a Prüfer encoding [12, 15] on problems of this type. Here we restrict our attention to more difficult problem types.

Correlated and Anti-Correlated

The correlated (and anti-correlated) graphs are generated by using the algorithm given in Figure 1. The procedure takes the required correlation $\alpha \in [-1, 1]$ as an argument and returns a weight vector of K weights where the first component is drawn from a uniform distribution, and all subsequent weights are either positively or negatively correlated with respect to the *first* component, and lie within the same range of values. Note that since the correlation exists between the first and each other component of the weight vector, a correlation of $|\alpha|$ exists between all pairs of components $w^k, w^l \ k, l \in 2..K$. The correlation between the components of a weight vector affects the shape of the associated Pareto front of the MST problem of the graph. A zero correlation gives a smooth, convex Pareto front with a fairly constantly varying gradient along its length. In contrast, a large positive correlation gives a convex Pareto front with more of a discontinuous change in the gradient. With a correlation approaching +1, the front becomes smaller until in the limit, it will only contain one optimal point. With a strong negative correlation, the front is convex but approaches a straight line or flat surface in objective space as the correlation approaches -1. Because of this there tend to be a large number of non-supported efficient solutions (those that do not lie on the convex hull of the Pareto front). This shape of Pareto front might make it difficult for methods based on the use of weighted sum aggregation of objectives, to find a good approximation to the Pareto front since they tend to find it difficult to discover non-supported solutions, and also rely on a changing gradient in the Pareto front to find a good range of points on it.

M-Correlated

The M-Correlated graph generator is based on a graph generator developed by us [6] for producing 'misleading' or M-graph problems for the standard (single-objective) d-MST problem, and combining this with the correlated graph generator described above. The M-Correlated graphs are designed to be particularly difficult to solve when a low maximum vertex degree constraint must be satisfied.

Algorithm: *Gen_correlated_wts*

$\alpha \in [-1, 1]$ is the correlation, provided by the user
β is the offset, calculated from α
γ is the variation, calculated from α and β
$\mathcal{U}(min, max)$ is a uniformly distributed random deviate $\in [min, max)$

```
if ($\alpha \geq 0$)
  $\beta \leftarrow 1/2(1 - \alpha)$
  $\gamma \leftarrow \beta$
else
  $\beta \leftarrow 1/2(1 + \alpha)$
  $\gamma \leftarrow \beta - \alpha$
foreach edge $(i, j) \in E$
  $w^1_{i,j} \leftarrow \mathcal{U}(0, 1)$
  foreach objective $k \in 2..K$
    $w^k_{i,j} \leftarrow \alpha w^1_{i,j} + \beta + \gamma \mathcal{U}(-1, 1)$
```

Figure 1: An algorithm for generating a graph with correlated weight vectors

In theory, the maximum vertex degree of a MST in a graph of random edge weights is $|V| - 1$. However, in practice, when reasonably large uniformly random weight graphs are generated, the maximum vertex degree of the graph's MST rarely exceeds four or five. Due to this fact, some researchers [1, 10] have developed methods for generating biased random graphs where the graphs' MSTs have a high maximum vertex degree. Knowles and Corne [6] further developed the graph generator of Boldon *et al.* to bias the edge weights in such a way as to mislead any algorithm that greedily chooses edges of low weight in an attempt to grow a low-weight spanning tree. The M-graph generator, as it is called, requires four parameters to be set:

$\lvert V\rvert$	the number of vertices in the graph;
f	the number of vertices with large degree;
ld	the lower bound on the degree of large-degree vertices; and
ud	the upper bound on the degree of large-degree vertices,

with the constraints that $f.ud < |V|$ and $ld < ud$. The generator has two main stages. In the first stage a spanning tree that will be the MST of the graph is formed. In the second stage other edges are added to form a graph of the required density. The first stage begins by forming f different 'star' graphs with the degree of each star centre vertex chosen uniformly at random in $[ld, ud]$. These disconnected components are then connected by adding $f - 1$ edges at random, to form a tree. The set of connected vertices, V_T, in the tree will have fewer than $|V|$ members, so that the tree is not spanning. To span the whole graph, additional vertices in $V \setminus V_T$, not in any of the stars, will be added. However, first the edges in the (non-spanning) tree formed so far, are all assigned uniformly random weights in $[0, \chi)$. Next, all remaining unconnected vertices are connected to the tree by adding an edge between them and exactly one of the star centre vertices. The weight of these edges is assigned a uniformly random weight in $[\phi, \chi)$. Additional edges are then added between any pair of non-adjacent vertices, until the graph reaches the required density of connectedness. The weights assigned to these additional edges are uniformly random in $[\chi, \omega]$ for any pair of vertices where both are members of V_T, and uniformly random in $[\psi, \omega]$ for all other vertex pairs. If the weight parameters are set so that $0 < \phi < \chi \ll \psi < \omega$ then the resulting graph will be a misleading graph, that is the graph's structure will successfully mislead algorithms that favour choosing low-weight edges. This can be understood by first noticing that the edges incident to the vertices in $V \setminus V_T$ have two different ranges of values. Those edges whose other incident vertex is a star-centre have a low weight, whereas all others have a high weight. However, often a greedy-style algorithm will be forced to choose the higher weighted edge to connect these vertices because in earlier choices it will favour the edges that are incident to a star centre vertex (because these have the lowest range of weights in the graph) thereby causing the star centre vertex to reach its maximum allowed degree, and so preventing the connection of it to one of the vertices in $V \setminus V_T$. Because $\psi \gg \chi$ this will lead to a heavier graph, overall.

To make a multiobjective version of an M-graph, we use the M-graph procedure to set the weights of the first component of all the edge weight vectors. The others components of the edge weight vectors are then set using the correlation procedure outlined above. If a large positive correlation is used then the graph will be misleading in all of its components, and it will be difficult for a greedy approach to find a low-weight solution if the degree constraint is much lower than the parameter ud.

Concave

The Concave problem generator can only be used to make bi-objective problems at present. It works by setting the edge weights of three 'special' vertices (labelled 1, 2, and 3) in such a way that a large concave region in the Pareto front will result. If we restrict all edge weights to lie in [0,1], then the weights that can be used are the following: $\mathbf{W}_{0,1} = (\zeta, \zeta)$, $\mathbf{W}_{0,2} = (0, 1-\zeta)$, and $\mathbf{W}_{1,2} = (1-\zeta, 0)$. All other edges are $\mathbf{W}_{i,j} = (\mathcal{U}(\zeta,\eta), \mathcal{U}(\zeta,\eta))$ for $i,j > 3$ and $\mathbf{W}_{i,j} = (\mathcal{U}(1-\zeta,1), \mathcal{U}(1-\zeta,1))$ if i xor $j \leq 3$, with $i,j \in V$, ζ a small positive value of the order of $1/|V|$, $\zeta < \eta \ll 1-\zeta$, and $\mathcal{U}(\min,\max)$ giving a uniformly random deviate in $[\min, \max]$.

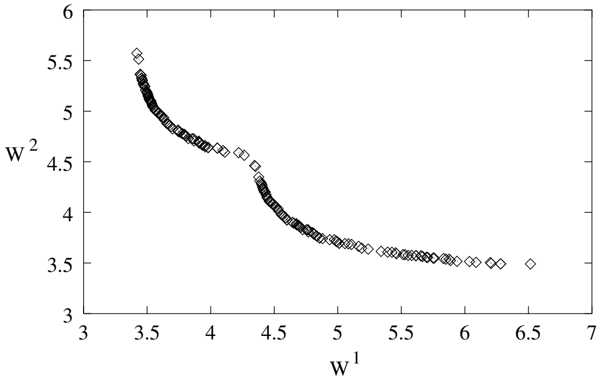


Figure 2: A plot showing true nondominated points on a 25 vertex concave problem

An example of (an approximation of) the Pareto front of a concave graph problem is given in Figure 2. The graph has 25 vertices and the values of the parameters for generating it were: $\zeta = 0.05$ and $\eta = 0.2$.

4 Algorithms

In a recently submitted paper [9], we studied the performance of two different encodings within a multiobjective evolutionary algorithm called AESSEA, on the mc-MST problem without degree-constraints. In that work, it was found that a mixed method based on a decoder-style encoding for initialization, and a direct encoding with specialized crossover and mutation operators was superior to the Prüfer encoding, in the same evolutionary algorithm. We also concluded in the paper that for un-constrained random-weight problems, a simple approach based on iterating Prim's algorithm [11] for different weighted-sum aggregations of the objectives in the given mc-MST, actually produces results that are far better than the Prüfer encoded evolutionary algorithm, and about the same as the mixed encoded evolutionary algorithm, but in far less time. However, we also noted that for various constrained problems and for situations where the Pareto front may contain many non-supported solutions, the Prim's algorithm approach would fail to perform well.

Our aim here is to test those ideas by applying AESSEA with a mixed encoding to a range of degree-constrained problems with different shaped Pareto fronts and once again we compare our results with our iterated mc$d$-Prim algorithm procedure, and also against a complete enumeration of the space on the smaller 10-vertex problems. The algorithms mc$d$-Prim and AESSEA are described briefly below, including the encodings and operators used in the latter.

Algorithm mc$d$-Prim

Prim's algorithm [11] is a well-known polynomial time constructive algorithm for solving the (single-objective, unconstrained) MST problem. It can be adapted to the d-MST by changing it so that at each step, in its construction of a tree, it checks for a degree-constraint violation before adding in the next edge. We call this constrained version of Prim's algorithm, d-Prim. Of course, since d-MST is NP-hard, d-Prim does not guarantee an optimal solution.

For tackling the mc$d$-MST, a multiobjective version of d-Prim is easily devised. By simply replacing the vector of edge weights in the graph by a weighted sum scalarization of them, optimization can be carried out in one 'direction' of the objective space. In mc$d$-Prim, this procedure is iterated for many different weightings of the objectives, giving a whole range of solutions approximating the Pareto front.

The scalarization of the objectives is achieved by replacing the vector weights defined on each edge in G by a scalar weight b formed by taking the inner product of a normalized scalarizing vector, $\boldsymbol{\lambda}$, and $\mathbf{w}$, $b = \boldsymbol{\lambda}.\mathbf{w}$. To obtain different weightings, we use every normalized scalarizing vector, $\boldsymbol{\lambda}$, with components equal to $l/s, l = 0..s$ where s is a parameter controlling the number of different vectors that will be generated. This gives $\binom{s+K-1}{K-1}$ different, evenly distributed, scalarizing vectors. For each different $\boldsymbol{\lambda}$ vector, the constrained version of Prim's algorithm (d-Prim) is applied.

For a large number of different scalarizing vectors (in our experiments we set $s = 1000$, giving 1001 different $\boldsymbol{\lambda}$ vectors), mc$d$-Prim may generate an approximation to S^*, the set of Pareto optimal spanning trees, that is satisfactory in many cases. However, with a low degree constraint the algorithm may not generate many (or any) optimal solutions on a single run. Therefore, it may be useful to run the algorithm several times with a different start vertex. In the experiments reported below, we always run mc$d$-Prim five times, each with a different start vertex.

AESSEA

The multiobjective evolutionary algorithm, AESSEA, is based closely on procedures already defined for the Pareto archived evolution strategy (PAES) [7], and is described more fully in [9]. AESSEA is a steady-state EA, that is, only one new solution is evaluated per 'generation'. It keeps a set of non-dominated solutions in an archive, and uses this set of solutions to estimate the quality of newly generated solutions. The algorithm is elitist in the sense that parents and offspring compete, but the overall selection pressure of the algorithm is not too strong since selection for mating is purely random, and offspring only replace one of their parents, rather than the weakest member of the population. Some testing of this algorithm and comparison with PESA [2] suggest that it is both an effective and computationally efficient, multiobjective EA.

RPM decoder encoding

The randomized primal method was put forward in [6] (where it is described in detail) as an encoding for solving the d-MST problem using any metaheuristic search method. It is a decoder type of representation, that is the chromosome encodes for choices that are made when a constructive algorithm builds a tree. The problem with this type of encoding is that it does not exhibit good locality, and it has super-linear growth in complexity for linear increase in the graph size $|V|$. However, it is good for initialization where it used only a small number of times.

Direct encoding and operators

The direct encoding and operators used in AESSEA are multiobjective versions of those put forward by Raidl in [13] for the d-MST problem. We describe how these operators were adapted for the mc$d$-MST problem in detail in [9]. In summary, the operators are adapted so that they bias the choice of edges towards those that are the minima on some weighted-sum single objective evaluation of the multiobjective tree weight. The weights in the weighted-sum are also encoded for by the chromosome and are subject to mutation and crossover. The method ensures that good solutions are found across the whole Pareto front.

5 Experimental method

Generated problems

Three graphs for each of the problem types: Correlated, Anti-Correlated, M-Correlated, and Concave were generated; one each at sizes of 10, 25, and 50 vertices, giving 12 graphs in all, from which 15 problems are created by setting degree constraints of 3 on all of the problems, and an additional, lighter degree constraint of 5 on the three M-correlated problems.

The correlations for the Anti-Correlated graphs, 10vAC, 25vAC, and 50vAC were set at -0.7. For the Correlated graphs, 10vC, 25vC, and 50vC, the correlation was set at 0.7. For the M-Correlated graphs the correlation was also set at 0.7, and the other parameters were $f = 1, 2, 5$, $ld = 6, 6, 7$, $ud = 8, 10, 9$, for the 10vM-C, 25vM-C, and 50vM-C, respectively. There is no correlation between the edge weight components in the concave graphs, and the other parameters used for generating these graphs were $\zeta = 0.1, 0.05, 0.03$ and $\eta = 0.25, 0.2, 0.125$ for the 10vConc, 25vConc and 50vConc graphs respectively.

Algorithm parameters and experiments

The parameters used for AESSEA are given in Table 1. AESSEA is run 30 times independently on each of the 15 problems and the nondominated archive returned by it from each run is stored for statistical analysis, and comparison with the non-EA approaches. For each problem, mc*d*-Prim is run 5 times with a different start vertex, and the combined nondominated solution set is stored. On the ten-vertex problems we also use an enumerative procedure to give us the entire true Pareto front for comparison.

Parameter	AESSEA
population size, $\|P\|$	200
nondominated solutions archive size, *arcsize*	200
initialization method	RPM
mutation type	edge-mutation
crossover type	edge-crossover
total number of function evaluations, *num_evals*	20k/50k/50k
# of grid squares used for 'crowding' strategy [7]	1024

Table 1: Parameter settings for AESSEA. The three figures for number of evaluations relate to the three different problem sizes, 10, 25, and 50 vertices, respectively

Statistical Analysis of Data

When a multiobjective EA or other approximate method is run on a multiobjective problem, it returns a set of (mutually nondominated) solutions that approximate the true Pareto optimal set. Each of the solutions also has an image in the multi-dimensional objective space, consisting of an objective vector. The set of objective vectors approximate the true Pareto front. How well the discovered objective vectors (points) approximate the true Pareto front is usually more important than the proportion of the Pareto optimal solutions that have been found. But measuring the quality of an approximation to the Pareto front is not a straightforward problem because several dimensions of quality in an approximation can be identified. Coupled with this problem is the need for some statistical analysis of the performance of the (stochastic) approximate method over multiple runs, to provide useful summarising data for the expected performance of the algorithm.

Our methods rely on a technique developed by Fonseca and Fleming [4], and later implemented and extended by us [7]. The technique relies on the notion that the nondominated points from any approximation to a Pareto front define a surface (called the attainment surface), that divides up the objective space into a region that is dominated by the discovered nondominated points, and a region that is not dominated by them. Over multiple runs, an approximate algorithm will generate multiple different attainment surfaces. By sampling the distance of these surfaces from an origin at many different angles, one can obtain statistical information about the expected position of the attainment surface along each different angled direction (or sampling line). For example, one can calculate the median attainment surface, or the quartile attainment surfaces of an algorithm. One can also compare directly the *whole* distribution of positions of the attainment surfaces obtained from multiple runs of two or more different algorithms. This is particularly useful when comparing the performance of two or more EAs (e.g. see [7]). Here, where we just have bi-objective problems, our first method is to simply plot the median attainment surface of AESSEA and compare it with the combined nondominated points found by mc*d*-Prim.

The second method is quantitative. For a single algorithm's sets of nondominated points, the method first computes (a sampling of) the attainment surfaces as above, and then calculates the (sampled) median and quartile surfaces. Now, to convert these surfaces into a simple figure of merit, the size of the dominated region of the surfaces can be calculated. For a bi-objective minimization problem (as we have here) this is simply the area above and to the right of the attainment surface up to some bounding rectangle. How the bounding rectangle is best set is open to debate. Here we calculate the weight of the worst feasible solution (heaviest spanning tree) for each of the objectives in turn, giving us a point (z^1, z^2) that is used as the upper right corner of our bounding rectangle. (The points were calculated using Prim's algorithm set to maximize). In our results we report the absolute (unnormalized) size of the dominated region of the median surface, and (to get an idea of the variation over different independent runs), the interquartile dominated region size. One disadvantage of this approach is that concave regions of the Pareto front are under-represented in the statistics. This is seen in the results section where there is a seeming difference in the results shown by viewing the plots and by the numerical results, for the concave graph problems.

6 Results

Problem	Total size		Median (Interqt)
	Enum	mc*d*-Prim	AESSEA
10vAC	21.3837	20.4108	**21.357 (0.0032)**
25vAC		**246.256**	245.175 (1.428)
50vAC		**1240.27**	1224.07 (7.6)
10vC	36.0955	35.3919	**35.7672 (0)**
25vC		363.233	**363.329 (0.109)**
50vC		1868.59	**1869.3 (3.52)**
10vM-C-*d*3	26.8362	23.6895	**26.8342 (0)**
25vM-C-*d*3		281.923	**343.892 (0.55)**
50vM-C-*d*3		1302.33	**1493.89 (3.17)**
10vM-C-*d*5	40.0389	35.3428	**40.0365 (0)**
25vM-C-*d*5		345.62	**384.511 (0)**
50vM-C-*d*5		1496.8	**1615.49 (4.56)**
10vConc	37.3255	37.0848	**37.2367 (0)**
25vConc		**334.694**	334.644 (0.354)
50vConc		**2122**	2118.39 (1.67)

Table 2: Size of the median and interquartile dominated regions for 30 runs of AESSEA, and the total combined size of the dominated region found using five runs of mc*d*-Prim. For the ten-vertex problems the true size of the dominated region is represented by the results of the Enumeration algorithm

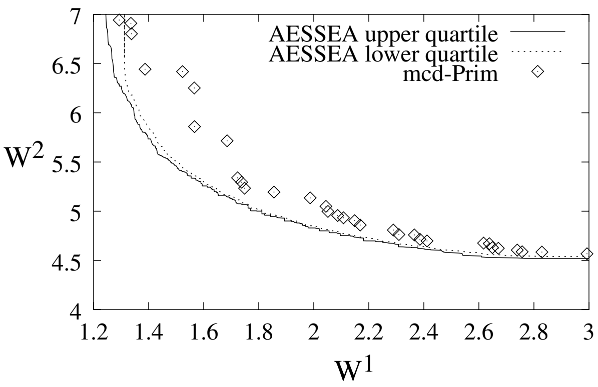


Figure 3: AESSEA's quartile attainment surfaces, and mc*d*-Prim's combined discovered vectors, on the 50 vertex correlated problem with degree constraint 3

From the results given in Table 2, we can see that mc*d*-Prim performs very well compared to AESSEA on the correlated and anti-correlated problems, and is considerably faster (but see Figure 3 for further help visualizing the Pareto fronts discovered). However, on the M-correlated graphs it clearly struggles. This is as expected because on these problems the degree constraint has a real effect on the difficulty of finding good solutions. We can see that AESSEA is clearly superior here; observe the size of the region discovered by it, compared to the enumeration method on the 10 vertex M-correlated problem for both d=3 and d=5. This is further shown in a plot given in Figure 4. Finally, on the (larger) concave problems, it appears that mc*d*-Prim does better than the AESSEA but in fact is unable to find any solutions in the concave region of the Pareto front. Its larger dominated region is due to it finding the very edge of the Pareto Front, which the EAs do not achieve on every run. This is illustrated in a plot showing the median attainment surface for AESSEA, and the points found from 5 runs of mc*d*-Prim is given in Figure 5.

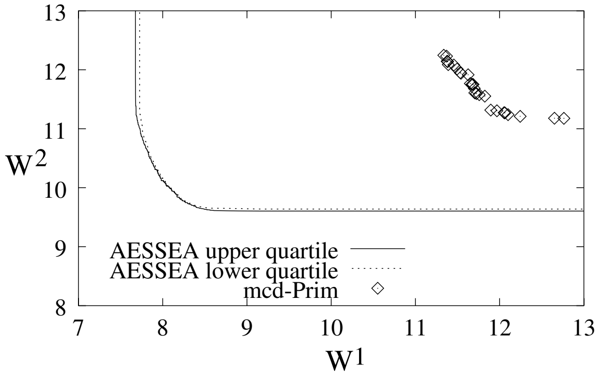


Figure 4: AESSEA's quartile attainment surfaces, and mc*d*-Prim's combined discovered vectors, on the 50 vertex M-correlated problem with degree constraint 3

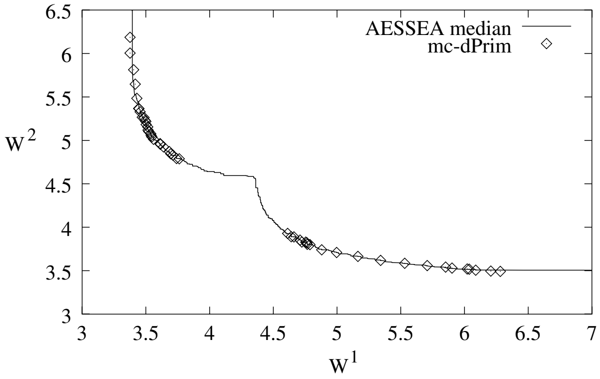


Figure 5: Nondominated points found from 5 runs of mc*d*-Prim, and the median attainment surface achieved by AESSEA. Note how AESSEA finds points in the concave region of the Pareto front

7 Conclusion

We have presented a number of graph generators that can produce a range of challenging graph types for the mc*d*-MST problem. The generators can be used to make problems for the OR community to test exact and heuristic approaches to the mc-MST, and for generating benchmark problems to form part of broader test problem suites for evaluating multiobjective EAs.

We have shown that on some problems it may not be necessary to use an evolutionary algorithm or other metaheuristic technique for tackling these problems, because a simple, iterative approach — m*cd*-Prim — can provide very good results in a fraction of the time. However, we have also demonstrated that for certain problems with constraints that are difficult to meet, an evolutionary algorithm, AESSEA, does obtain superior results. Furthermore, AESSEA is able to find points in the non-supported regions of the Pareto front, as was clearly demonstrated using the concave graph generator. In real problems of interest to the telecommunications industry, the number and variety of constraints that must be met will necessitate the use of evolutionary algorithms similar to that investigated here.

In our future work we will investigate the performance of a memetic algorithm, M-PAES [8], on these problems. Although M-PAES has been shown to perform well on other problems, it seems likely that in this application, a local-search element would be particularly useful. We base this conjecture on the observation made by Ehrgott and Klamroth [3] that from a sample of 50 random instances of a random weight bi-objective MST problem, all of them were ergodic with respect to a single exchange operator (although they prove this will not always be true). In light of this, we predict that further advances in tackling these difficult constrained mc-MST problems will come from techniques that incorporate a strong local search element, as used in M-PAES.

Acknowledgments

The authors would like to acknowledge the continuing sponsorship of British Telecommunications Plc., and thank the anonymous reviewers for useful comments.

References

[1] B. Boldon, N. Deo, and N. Kumar. Minimum-Weight Degree-Constrained Spanning Tree Problem: Heuristics and Implementation on an SIMD Parallel Machine. Technical Report CS-TR-95-02, Department of Computer Science, University of Central Florida, Orlando, FL 32816, 1995.

[2] D. W. Corne and J. D. Knowles. The Pareto-envelope based selection algorithm for multiobjective optimization. In *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN VI)*, pages 839–848, Berlin, 2000. Springer-Verlag.

[3] M. Ehrgott and K. Klamroth. Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal of Operational Research*, 97:159–166, 1997.

[4] C. M. Fonseca and P. J. Fleming. On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature—PPSN IV*, Lecture Notes in Computer Science, pages 584–593, Berlin, Germany, September 1996. Springer-Verlag.

[5] H. W. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52:209–230, 1994.

[6] J. Knowles and D. Corne. A new evolutionary approach to the degree-constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4(2):125–134, July 2000.

[7] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[8] J. D. Knowles and D. W. Corne. M-PAES: A Memetic Algorithm for Multiobjective Optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC00)*, pages 325–332, Piscataway, NJ, 2000. IEEE Press.

[9] J. D. Knowles and D. W. Corne. A comparison of encodings and algorithms for multiobjective minimum spanning tree problems. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC'01)*. IEEE Press, 2001. (To appear).

[10] M. Krishnamoorthy, A. T. Ernst, and Y. M. Sharaiha. Comparison of algorithms for the degree constrained minimum spanning tree. Technical report, CSIRO Mathematical and Information Sciences, Clayton, Australia, 1999.

[11] R. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36:1389–1401, 1957.

[12] H. Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys.*, 27:742–744, 1918.

[13] G. R. Raidl. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC00)*, pages 104–111, Piscataway, NJ, 2000. IEEE Press.

[14] R. M. Ramos, S. Alonso, J. Sicilia, and C. González. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111:617–628, 1998.

[15] G. Zhou and M. Gen. Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. *European Journal of Operational Research*, 114(1), April 1999.

[16] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

Emergence of Profitable Search Strategies Based on a Simple Inheritance Mechanism

Natalio Krasnogor
Intelligent Computer System Centre
University of the West of England
Bristol, United Kingdom
`Natalio2.Krasnogor@uwe.ac.uk`
`http://www.csm.uwe.ac.uk/~n2krasno`

Jim Smith
Intelligent Computer System Centre
University of the West of England
Bristol, United Kingdom
`James.Smith@uwe.ac.uk`
+44 117 3443357

Abstract

In this paper we show how a simple inheritance mechanism is capable of learning the best local search to use at different stages of the search. In our work an individual is composed by its genetic material and its memetic material. The memetic material specifies the strategy the individual will use to do local search in the vicinity of the solution encoded in its genetic part. A simple vertical inheritance mechanism is enough to provide a robust adaptation of behavior. This result spans from a simple *OneMax* problem, to *NK-landscapes* and the *TSP*.

1 INTRODUCTION

In this paper we introduce a Memetic Algorithm (MA) in which the local search (a meme) employed by each individual is learnt during evolution. An individual is composed of its genetic material and its memetic material. The memetic material specifies the strategy the individual will use to do local search in the vicinity of the solution encoded in its genetic part. A simple vertical inheritance mechanism, as used in self-adaptive genetic algorithms and evolutionary strategies, is enough to provide a robust adaptation of behavior. We begin by illustrating the viability of the adaptive mechanism with two experiments where the GA adapts to use suitable mutation probabilities. With our method any meme, that is any mutation rate, is accesible with equal probability from any other one. This can not be achieved by a binary or gray encoding using multiple bits[1][17] nor with a real value encoding attached to the normal genes [8]. Also, by using this mechanism, the control of which memes to use is a distributed one. Furthermore, memes themselves can be modified by an adequate mechanism. For a detailed review of operator adaptation refer to [18]. This is then expanded to a MA where the memes represent local search algorithms. In the memetic algorithms literature authors have spent a considerable amount of research assessing, e.g., how deep the local search should be and how often[7]. Land[13] used the concept of “sniffs” to try to gauge which individuals should go through a local search phase and with how much intensity. In [6] the authors developed a systemic model of Global-Local search hybrids that shed some light on the optimization of those algorithms. Carrizo et.al. in [5] employed several local searchers within the same MA to solve quadratic assignment problems. Moreover, to the best of our knowledge, just a few papers[19][10] have appeared where the choice of **which** local search to apply was left to the evolutionary process itself. It is in this spirit that this work is done.

2 THE MEMETIC ALGORITHM AND THE SIMPLE INHERITANCE MECHANISM

In this section we will describe the underlying GA architecture used in our experiments. An individual is composed of genetic material plus a meme allele.The genetic part was the representation of the potential solution. There were M memes available to be expressed by an individual, that is to say we treat our memes as categorical rather than ordinal entities. For the *One-Max* and *NK-Landscapes* Problems memes represented mutation strategies. In this case they were not associated with any local search process so we can regard our memetic algorithms as an adaptive GA. In the case of the *TSP*, memes were chosen from a range of local search strategies, embodying a fully fledged MA. The mutation process of an individual involves mutating its meme and its chromosome. The meme is mutated accordingly to a small innovation_rate IR by randomly choosing a meme number from the distribution $U(1, M)$. The IR takes a value in the range $[0, 1]$. A value of 0 means that there is no innovation and hence if a meme allele is lost it will not be re-introduced in the population. A value of 1 specifies an extremely explorative meme policy where all the different strategies implied by the available M memes will be equally used and no emergent properties are expected to arise. After that, the mutation strategy given by the meme is expressed. This mutation strategy specifies the kind of genetic mutation (One Point mutation or Bit Wise mutation) and the probability of applying it to the chromosome. The innovation_rate guarantees a minimum level of exploration of the memetic space. For an innovation_rate of IR, a population size of μ and a uniform distribution of meme mutations $U(1, M)$, even the worst meme can be reintroduced to the population with a frequency of $P_r = \frac{IR \times \mu}{M}$ per generation. Crossover is based on the following pseudocode:

```
Individual_Level_Crossover(parent1, parent2)
BEGIN
 IF(both parents carrie the same meme}
  Cross parents genetic material.
  Inherit common meme to offspring.
 ELSE-IF (parent1.fitness()==parent2.fitness())
  /* the two parents have different memes          */
  /* but their fitness are comparable hence         */
  /* a random choice is made                        */
  Cross parents genetic material.
  Choose a meme randomly from any of the two parents.
  Inherit selected meme to offspring.
 ELSE
  /* parents don't share memes nor fitness values   */
  /* hence the fittest individual                   */
  /* imposes  its meme preference                   */
  Cross parents genetic material.
  Choose meme from fittest parent.
  Inherit the chosen meme to offspring.
END
```

The first phase involves the standard chromosome crossover, while the second phase performs the vertical propagation of the memes in the following way. If two individuals share the same meme then this meme will be inherited to the offspring. If the memes they carry are different, then the meme of the fittest parent is propagated. Finally, if memes are different but the fitnesses are equal, then a random choice between both memes will be done and the selected one will appear in the offspring. The memetic phase of the crossover is kept identical in the three problems. The first phase of chromosome crossover are different; for the *OneMax* uniform crossover with probability 0.7 was used, for the *TSP* DPX with probability 0.6 was used. In the case of *NK-Landscapes* no chromosome crossover was employed. As we said before an individual consists of its chromosome and its meme. This meme specifies a strategy that is composed of both an operator and its probability of being applied. These composed memes are called 'memeplexes' (see for example [3]). In the first two experiments presented the meme encoded a fixed mutation operator with variable probabilities for the binary problems, while for the *TSP* the meme represented variable local searchers with a probability of being applied fixed at 1.0 .

3 EVOLUTIONARY ACTIVITY WAVE AND MEME CONCENTRATION GRAPHS

In order to examine the evolutionary and adaptive properties of memes in our system we will use the approach of Bedau et.al. [2]. We are interesting in observing the adaptive significance of the search strategies coded by the memes. We define the *concentration* c of meme i at time t as the number of individuals in the population that carry this meme. We denote this value by $c_i(t)$. It is hypothesized that since memes are carried alongside genes, those strategies that confer a selective advantage to the genes (i.e. they represent an efficient local search) will proliferate. Moreover, this proliferation is going to be reflected as an increase on those meme's concentration. The meme concentration $c_i(t)$ is a crude measurement of a meme success because it doesn't give information about its continual usage since it first appeared in the population. Some memes might have a low concentration at a given time but disappear or take over the population in the next few generations. To account for such phenomena the *evolutionary activity (E.A.)* a of meme i at time t is defined by:

$$a_i(t) = \begin{cases} \int_0^t c_i(t)dt & \text{if } c_i(t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

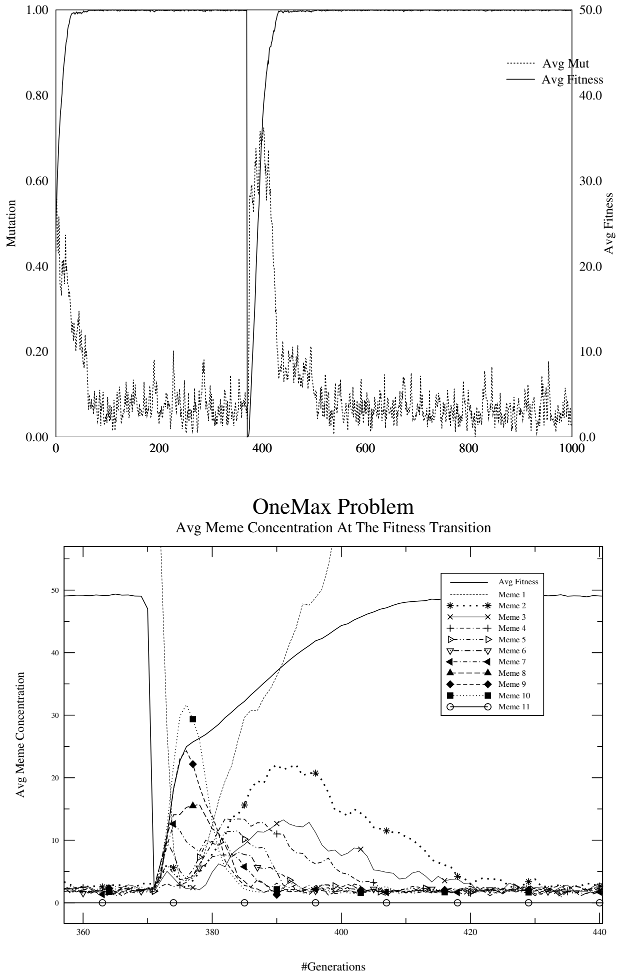
When the evolutionary activity of a meme is plotted, the slope of the curve represents the concentration $c_i(t)$ of meme i. In that way, an increase in the slope indicates an increasing use of a given meme (it is spreading fast in the population). An almost flat curve points out a meme that is having less survival value than its competitors. In this kind of graph is usual to distinguish a wave of activity when a meme that has been successful for many generations disappears.

4 THE *OneMax* AND *NK-Landscapes*

In this section we describe several experiments performed to understand the behavior and the feasibility of adapting memes in a population of evolving individuals. We will describe and analyze the results of several experiments on two different, yet related, problems: *OneMax Problem* and *NK-Landscapes*[9]. The *OneMax* problem consists of achieving an all ones bit string of length n starting from a randomly initialized population. The *NK-Landscapes* are binary problems of length n where genes participate in epistatic interactions. The number of genes with which any other gene interacts depends on K.

Furthermore, in the case of *OneMax*, in generation 370 (out of 1000) the problem was changed from maximizing the number of 'ones' to that of maximizing the number of 'zeroes'. This change in the fitness function provides a dynamic environment where individuals (genes and memes) were tested against very different situations. At the beginning of the run the population was randomly initialized, with both genes and memes set randomly. After the environment transition the evolving population was faced with a new problem (that of ZeroMax instead of ones). In practice this was equivalent to restarting the experiment but with a non-random population. In this way we were able to study the behavior of our approach under three different regimes: A random starting one and its adaptation towards an optima, a transient state with a biased (converged) initial population, and a final converged state. Thirty runs were made with a generational GA with no elitism. Deterministic binary tournament was used to select parents. The population size was 50. Uniform crossover was used with a probability of 0.7 . The 11 memeplexes specified a one point mutation together with its 'per individual' probability of being used. The mutation probabilities were in the range $[0.0, 1.0]$. For this experiment $IR = 0.1$. In figure 1 we plot the average of the mutation probabilities in the memes that exist at time t in the population. Also the average fitness achieved by the population is shown.

The dominating meme corresponds to a 0 mutation probability. This meme is successful because it preserves whatever the evolutionary system achieved. At the fitness transition it is wiped out and memes that represent high mutation rates take over the population (memes with mutation probabilities of 0.6, 0.8 and 0.9). When the new problem, ZeroMax, is 50% solved (i.e. half the allele values are optimal) those memes rapidly disappear and again strategies that represent low mutation rates dominate the population. From the graph in figure 1 we can see that at the beginning of the run, when the population is randomly initialized and the goal is to maximize the number of ones, the average mutation probability expressed by the memes is around 0.5. At the fitness transition and because the population is biased towards all ones, the system tunes to a much higher average mutation: 0.7 . When the system starts to maximize the number of ones on aver-



(b)

Figure 1: In (a) Average mutation rate expressed by the individuals in the population at time t. The mutation rate is defined by the meme each individual keeps, higher meme number means higher mutation rate. Average fitness is in solid line. Fitness transition is at generation 370. (b) Average meme concentration in the system at the fitness transition. Memes with higher numbers dominate (high mutation rates)

age half the bits will be properly set. After achieving an average fitness of at least 50%, mutations become deleterious and selection works against mutation. On the other hand, when the system switches to maximizing the number of zeroes, the majority of allele values are suboptimal, so with high probability mutating a bit will give a selective advantage. Thus, those memes which correspond to a higher probability of achieving this advantage will flourish until the mean density of zeroes is greater than 50%. Furthermore the expansion of the high mutation rates is longer than the one at the beginning of the run. In figure 1(b) the average meme concentration can be seen. The patterns of concentration at the start of the search and during the fitness transitions differ from each other (not shown here). During the fitness transition memes with even higher mutation rates are favored and it takes longer for their concentrations to decrease. This experiment demonstrates the ability of a system with simple meme encoding to adapt, even though (unlike other approaches) memes are not treated as continous or ordinal entities. A theoretical model of this method can be seen in [16].

5 ADAPTATION AND PHASE TRANSITIONS IN PARAMETER SPACE

In the previous sections we described the main architecture of our adaptive GA and we showed that it was capable of tracking changes in the environment by appropriately tuning the mutation rates of the evolutionary search. We also conducted a series of experiments with *NK-Landscapes* to assess if our system was able to adapt the mutation rates in more complex settings. In [14] the authors explore a phase change in search when a parameter τ reaches a certain critical value on some *NK-Landscape* problems . In their experiments the authors focused on Simulated Annealing (SA) as a local search algorithm, although a very special SA: the temperature was kept equal to zero at all times. The underlying operator, a bit-flip, was parameterized with τ, a per bit mutation rate. In their paper the authors show experimentally that the quality of the search follows an $s-$shape curve when plotted against τ making evident a change in phase. We wanted to explore whether the same kind of phenomenon arises in a GA and, if indeed this was the case, if our adaptive mechanism was able to select mutation rates comparable to those before the demeliorating transition.

As a first step we ran extensive simulations of the GA behavior with different bit rate mutations covering a wide range of values. The same GA as before was used but with a zero probability of crossover and 100 generations. A set of experiments was done with $N = 40$ and $K \in [0, 15]$. For each K three landscapes were created and 10 runs made on each landscape. This was repeated for 29 mutations rates in $\{0.0005, 0.0010, \ldots, 0.0045\} \bigcup \{0.005, 0.010, \ldots, 0.10\}$. In the upper part of figure 2 we can see the results obtained[1]. The GA is sensitive to the per bit mutation probability, there is a change in behavior at $\tau = 0.01$. When τ is further increased a swift loss of performance occurrs. This critical value τ_c is very close to the theoretically predicted error threshold[15] for finite asexual populations:

$$\tau_S^* = \frac{\ln \sigma}{\nu} - \frac{2 * \sqrt{\sigma - 1}}{\nu * \sqrt{S}} + \frac{2 * \ln \sigma * \sqrt{\sigma - 1}}{\nu^2 * \sqrt{S}} \qquad (2)$$

where S is the population size, ν is the genome length, σ is a "selective ratio" that gives a rough measure of fitness superiority of the master sequence. In our experiments $S = 50, \nu = N = 40$ and $\sigma = 2.0$, the resulting $\tau_S^* = 0.0103$.

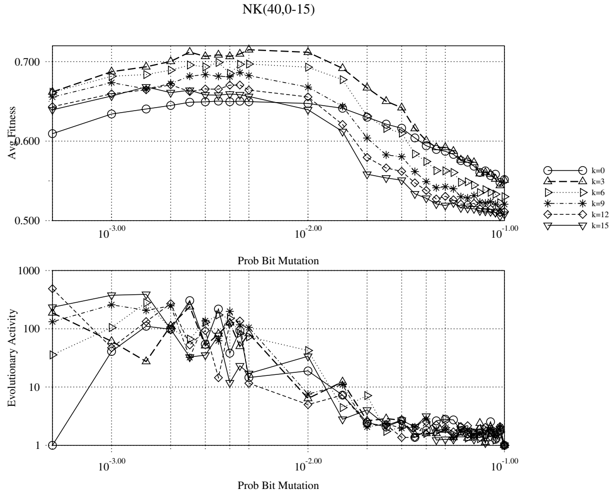


Figure 2: Up: Fitnesses achieved under different mutation rates for varying K averaged over 30 runs. Note the sharp decrease in fitness for mutations higher than 0.01. Down: Evolutionary activity of memes representing the same range of mutation probabilities as above. Note the fast decrease in activity for those memes above 0.01.

Once we knew that the phenomenon was present in GAs, we needed to check if the mechanism proposed here was capable of avoiding effective mutation rates equivalent to those greater than τ^*. In our experiment we used 50 individuals per generation, the GA was a

---

[1] For clarity we show here just a few K that span the range studied

generational one and the memes encoded per bit mutation rates in the range described above and a zero probability of crossover. The problems used were as in the exhaustive experiments. In the lower part of graph 2 we see the evolutionary activity of memes, as defined by equation 1, for K in the same range as before. The graph shows the activity for generation 100. Memes were associated with the range of probabilities with which the exhaustive runs were performed. We can see that the adapting GA was able to distinguish between memes before and after the τ^*. This is shown by the rapid decrease in evolutionary activity for those memes lying beyond 0.01. A second important conclusion that we can draw is that this simple adaptive mechanism is sensitive enough to be able to discriminate between a large set of alternatives (29 in this case) and it allows the emergence of effective mutation rates that avoid been trapped after τ^*. In figure 3 we plot, for different K, the best fitness obtained from all of the exhaustive runs of the standard GA, the fitness of the adaptive GA and the fitness of the standard GA after the transition. As it can be seen in the graph the adapting GA, by differentially propagating memes that are before and after the transition, can sustain fitness values comparable to the optimal ones. As K increases, the gap between the adaptive and the optimal value decreases, while the gap with the values after the transition gets larger. Another observation is that the transition is not sharp for $K \leq 4$, confirming Macready's et.al. findings [14]. Important to note is the fact that the best fitnesses obtained for the standard GA (before and after the transition) were obtained with different mutations rates and involved 30x29 runs. The adaptive version achieves its results just with 30 runs.

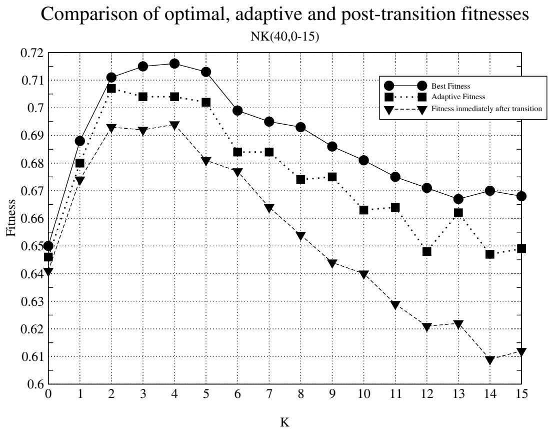


Figure 3: The best fitness for each K is compared to the fitness achieved by the adaptive GA and the fitness immediately after the transition at $\tau^* = 0.0103$. The values on this curves are averaged over 30 runs

6 ADAPTING THE BEST MEME FOR THE TSP

In the previous sections we showed that a simple vertical inheritance of memes was capable of performing an efficient adaptation of behavior for the dynamic *One-Max* and *NK-Landscapes*. In this part of the work we applied the same principle to learn which is the best meme to employ during different stages of the search for *TSP*. The *TSP* consists on finding the minimum length closed circuit among all the cities of a predefined set. The circuit should touch each city only once. We used 24 different memes, each meme defines the acceptance strategy, the underlying basic move and the number of iterations to use during the local search stage. There were two acceptance strategies, namely *first-improvement* and *best-improvement*. Three basic moves were considered $2-exchange$, $3-exchange$ and $4-exchange$. The final property of a meme was the number of times the acceptance strategy was going to be iterated employing the basic move. We can represent a meme M by the three values that specifies its basic move(M), its acceptance strategy(FB) and its number of iterations(I): $MeFBbIn$. The range of e was $\{2, 3, 4\}$ implying a $2-exchange$, $3-exchange$ or $4-exchange$. To specify a *first-improvement* acceptance strategy b was set to 1, and when $b = 2$ then the meme used a *best-improvement* acceptance strategy. Finally, n gives the number of iterations drawn from the set $\{1, 3, 6, 9\}$. Because our goal in this paper is to see if this simple memetic system can learn the best meme to use and not to discover the best meme for a particular *TSP* instance, we assume that the execution cost of all of the 24 memes is equivalent. The reader should note that the memes with $b = 2$ require greater computational cost than their counterparts $b = 1$. Furthermore, except for the $2-exchange$ (for which the neighborhood explored by the acceptance strategy was complete), just a sample of the induced neighborhood was considered for the other moves. For all the experiments run the probability of mutation was 0.4, that of crossover 0.6 and the innovation rate was set to 0.125. The crossover used was DPX and the mutation operator the double-bridge move. The underlying GA was a generational GA with a $(50, 200)$ strategy with a tournament size of 4. The architecture of the MA was, according to [11], a $D = 4$ MA, that is, local search was executed independently of mutation and crossover in a separate stage. The probability of local search (expressing the meme) was 1. The encoding used was a permutation encoding.

We first ran a set of experiments (one for each of the 24 memes), each consisting of 30 trials, where the whole population used the same meme, that was fixed during the complete run. The goal of this experiment was to obtain a ranking of memes for the different instances. We then ran an adaptive MA where the meme alleles were evolved. The graph in 4(a) shows the evolution of fitness over time for different memes on the lin318.tsp instance from TSPLIB. The reader must keep in mind that the memes were executed within the underlying MA described above.

An ANOVA analysis of the average over 30 runs for the best tour in each experiment shows that the curves are (with 95% confidence level) different. The ANOVA, together with the post-hoc $t - test$, provide a sound ranking of the various memes. As can be seen in figure 4(a) the MultiMeme MA, that is, the memetic algorithm for which the adapting process was enabled, was able to closely follow the performance of the best meme. It achieves this by favorably selecting the memes that produce the best increment in fitness. This is shown by the evolutionary activity graph in figure 4(b). The same results, with statistical significance, were obtained for other instances of different size and nature: eil76.tsp, lin105.tsp and mnpeano44.tsp. We ran extensive experiments with a MultiMeme MA where the memes available were of the form $MeFB1In$ with $e \in \{2 - exchange, 3 - exchange, 4 - exchange\}$ and $n \in \{1, 3, 6, 9\}$. The algorithm was able to positively select the best meme and to match the performance of the best one (with statistical significance). As the size of the instances increased the memes were more easily differentiated, and MultiMeme was able to track the curve of the best meme. For the instances studied the evolutionary activity diagrams show[2] that while the evolutionary search is not yet stagnated and the search is progressing, just one or two evolutionary waves are conspicuous, while the other memes remain under spurious activity. The use of an $IR > 0$ means that memes have non-zero background activity even if they are actually selected against (see section 2). When the search is converging towards local optima then several memes become neutral to each other and the evolutionary waves starts to develop. From our experiments it turns out that the best meme for all the instances was $M2FB1I9$ and the second best $M2FB1I6$. This is not surprising since those memes perform 9 and 6 iterations respectively based on the complete neighborhood of a $2 - exchange$ while for the remaining moves the neighborhood was sampled. This fact led us to design an experiment where the memes involved where the same as before **except** that the first best and the second best were not allowed to appear in the population. This MA will be called multiMeme-b. The results are shown in figure 5(a) for instance lin318.

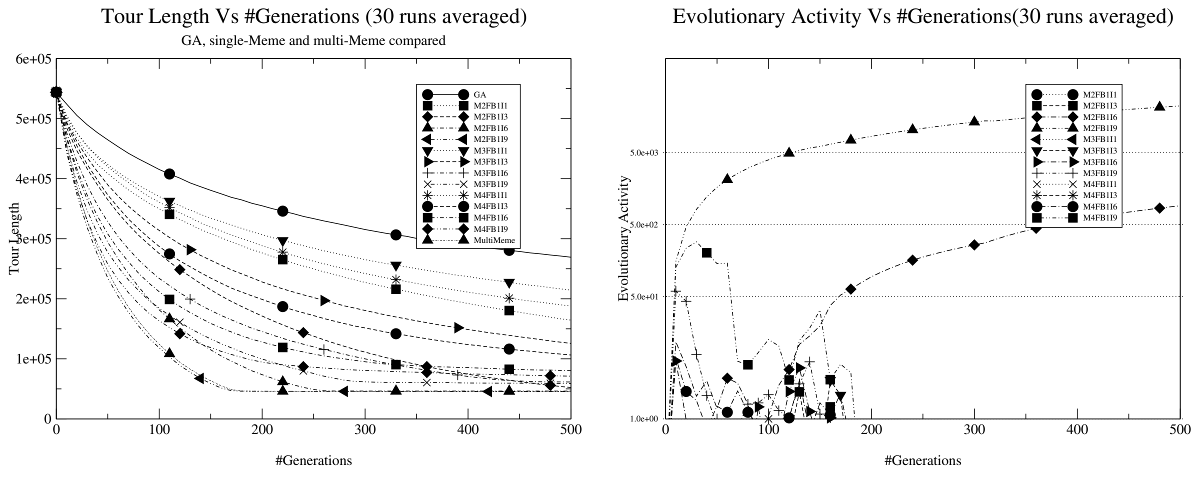
After analyzing the results obtained, we observed that multiMeme-b was able to track and follow the curve of the best meme for the instance mnpeano44.tsp, however it fails to do so for lin105.tsp and lin318.tsp. In the later case the best meme, that is, the meme that at the end of the run produces the best fitness was $M2FB1I3$. The algorithm fails to select this one in favor of $M4FB1I9, M3FB1I9, M3FB1I6$ and $M4FB1I6$. The reason for this behavior is simple to state: it pays for an individual to carry the meme that produces the maximum increase in fitness at any given point in time. Given that the individuals have no foresight of which is going to be the best fitness at the end of the run[3], the meme that produces the behavior with the steepest increase in fitness (decrease in tour length) is favorably selected. However, as generations go by, the relative payoff of the different memes change. In the case of the *TSP* the reason for this dynamic payoff is rooted in the so called "Big Valley" structure. In Boese's work [4] it is shown that the TSP shares with other commonly studied NP-Hard combinatorial optimization problems a globally convex structure of the set of local minima, where the local minima are points in the landscapes defined by different local search heuristics. The author shows that tours found by better heuristics are on average closer to each other in terms of distance[4] to the optimal solution, giving rise to the "Big Valley" metaphor. The gradient of improvement for the different memes changes during evolution while approaching a local optimum (eventually a global optimum). The adaptive MA, through its simple inheritance model is sensitive to this changes. Looking at the graph (b) in figure 5 we can see that the evolutionary wave of meme $M4FB1I9$ is becoming almost flat. As explained in the previous sections this means that the selective advantage of carrying this meme is decreasing. Also, we can see that waves of evolutionary activity arise for memes $M3FB1I9, M3FB1I6, M2FB1I3$. Tracing back the origins of these waves it is possible to note that they match the time when the corresponding curves in graph (a) to the left surpass the curve of multiMeme-b. Three vertical bars are marked in the graph with $x1, x2, x3$. Moreover, the longer the simulation, the closer the gap between the curve of the best meme and the multiMeme-b approach. The same behavior was notice for the instance lin105.tsp. The

---

[2] Only the graph for lin318.tsp is shown due to space limitations

[3] The system is not teleological.

[4] Distance here is actually measure as the number of links that differentiate two tours



(a) (b)

Figure 4: (a) Performance of different MAs when memes varies. Just the curves for the *first-improvement* strategy are shown. It is possible to see how the MultiMeme MA follows closely the performance of the best meme. In (b) the evolutionary activity of the MultiMeme MA is shown

reader should also note how the suppression of the best and second best memes alters the evolutionary activity diagram by comparing figure 4(b) and 5(b).

7 CONCLUSIONS

In this paper we showed how a simple vertical inheritance mechanism is enough to adapt the behavior of individuals in a memetic algorithm under different problems. Individuals have access to a set of memes that represent different search strategies. The evolutionary mechanism ensures that memes that are useful will be selected and spread in the population. Our approach differs from others (i.e. [19]) in that we are learning the association between an individual an a memeplex and not a vector with the particular characteristics of a given meme in a set of memes. Hence, the dimensionality of the problem is much smaller. From an engineering point of view this is a sound approach because we can allow memes to change using any algorithm that we find suitable, i.e., we can run a GA to define the memes themselves. By isolating the structure of a meme from its phenotypic action in the genes we are facilitiating the search in both genes and memes spaces. We tried our MA under three different scenarios. The dynamic *OneMax* problem showed that the adaptive MA was able to track changes in the environment, i.e. the fitness function, by triggering high mutation rates. We saw that for the *NK-Landscapes* the adaptive mechanism was robust enough to adapt to the edge of the transition after which mutation rates become pernicious. It was able to express an almost optimal mutation and to track closely the optimal fitness achieved by exhaustive runs. In the case of the *TSP* we are able to conclude that the adaptive MA is capable of selecting the memes that provide the best performance at any given moment of time. As a by product of this paper we were able to show that the phenomena described by Macready et.al. for simulated annealing [14] is also present in GAs. The memes exploited in this paper were, accordingly to [11], static memes. We are currently running experiments with adaptive memes like those of [12]. Self adapting memes for the Protein Folding Problem will be studied soon.

References

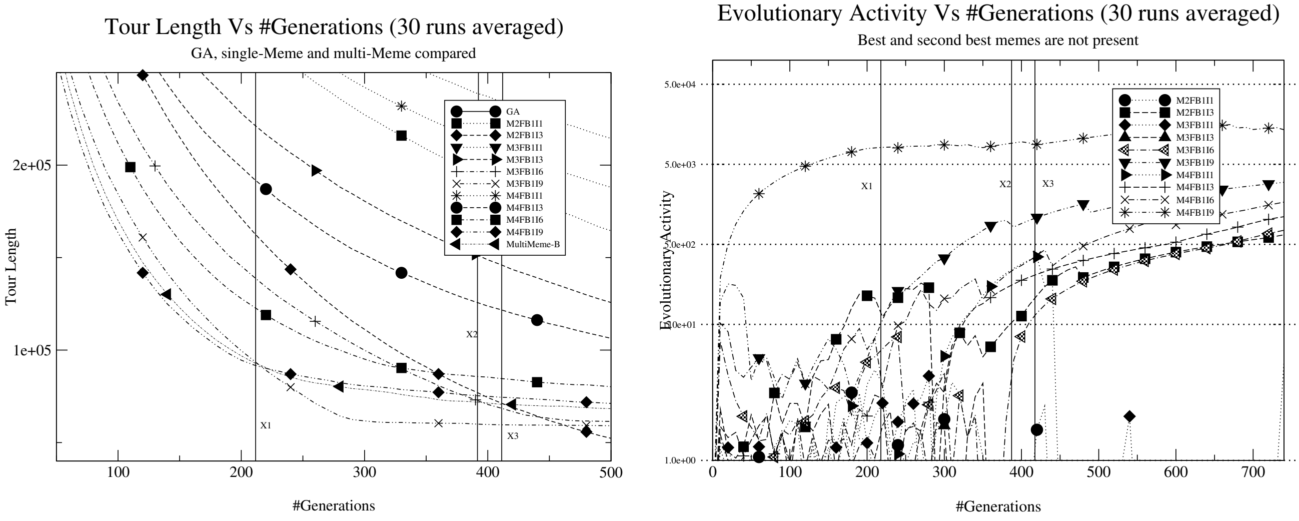
[1] T. Back. Self adaptation in genetic algorithms. In F. Varela and P. Bourgine, editors, *Towards a Practice on Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 263–271. MIT Press, 1992.

[2] M. Bedau and N.H.Packard. *Measurement of Evolutionary Activity, Teleology and Life*, volume 98-03-023. Addison-Wesley, 1992.

[3] S. Blackmore. *The Meme Machine.* Oxford University Press, 1999.

[4] K. Boese. *Models For Iterative Global Optimization.* Ph.D. Thesis, UCLA Computer Science Department, 1986.

[5] J. Carrizo, F. Tinetti, and P. Moscato. A computational ecology for the quadratic assignment problem. In *Proceedings of the 21st Meeting*



(a) (b)

Figure 5: (a)Zoom-in in the performance graph. MultiMeme-b is a MA with all the memes available except the first best and second best. In (b) the evolutionary activity of the MultiMeme-B MA is shown

on Informatics and Operations Research, Buenos Aires, 1992. SADIO.

[6] D. Goldberg and S. Voessner. Optimizing global-local search hybrids. In W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakaiela, and R. Smith, editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 1999.

[7] W. E. Hart. Adaptive global optimization with local search. *Ph.D. Thesis, University of California, San Diego*, 1994.

[8] R. Hinterding, Z. Michalewicz, and T. Peachey. Self-adaptive genetic algorithms for numeric functions. In *Proceedings of Parallel Problem Solving from Natur - PPSN IV*, 1996.

[9] S. Kauffman. *The Origins of Order, Self Organization and Selection in Evolution.* Oxford University Press, 1993.

[10] N. Krasnogor. Co-evolution of genes and memes in memetic algorithms. In A. Wu, editor, *Proceedings of the 1999 Genetic And Evolutionary Computation Conference Workshop Program*, 1999.

[11] N. Krasnogor and J. Smith. Memetic algorithms: Syntactic model and taxonomy. submitted to The Journal of Heuristics. Available from the authors.

[12] N. Krasnogor and J. Smith. A memetic algorithm with self-adaptive local search: Tsp as a case study. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2000.

[13] M. Land. Evolutionary algorithms with local search for combinatorial optimization. *Ph.D. Thesis, University of California, San Diego*, 1998.

[14] W. Macready, A. Siapas, and S. Kauffman. Criticality and parallelism in combinatorial optimization. 1995. to appear in Science.

[15] G. Ochoa and I. Harvey. Recombination and error thresholds in finite populations. *Foundations of Genetic Algorithms - 5*, 1999.

[16] J. Smith. Modelling gas with self adaptive mutation rates. In *GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufman, 2001.

[17] 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.

[18] J. Smith and T. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, pages 81–87, 1997.

[19] H. Terashima-Marin, P. Ross, and M. Valenzuela-Rendon. Evolution of constraint satisfaction strategies in examination timetabling. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

Evolutionary Algorithms in Control Optimization: The Greenhouse Problem

Thiemo Krink
EVALife Group
Dept. of Computer Science
University of Aarhus, Denmark
krink@daimi.au.dk

Rasmus K. Ursem
EVALife Group
Dept. of Computer Science
University of Aarhus, Denmark
ursem@daimi.au.dk

Bogdan Filipič
Dept. of Intelligent Systems
Jožef Stefan Institute
Ljubljana, Slovenia
bogdan.filipic@ijs.si

Abstract

In recent years, evolutionary algorithms (EAs) have been applied to a variety of dynamic optimization problems. In control optimization, dynamic problems are characteristically dominated by the feedback between the controller and the controlled system. Most studies in this field are rather pragmatic and many principal issues in control optimization have not been addressed yet. In this paper, we studied the performance of various control strategies regarding the investment of computation time in number of generations versus population size. Further, we investigated the evaluation of candidate solutions in respect to their changing fitness over time. Our experiments showed that both aspects were significant factors for the optimization performance. As a benchmark control problem, we implemented a simplified model of a crop producing greenhouse, where the objective was to maximize the profit. The fitness landscapes generated with this simulator showed that previously suggested test case generators cannot model realistic control problems.

1 INTRODUCTION

The ultimate goal of an optimization technique is its application to real-world problems. Typically, real-world optimization problems are characterized by constraints, multiple objectives, and dynamic properties. In particular control problems are notoriously dynamic due to the feedback between the controller and the controlled system. Therefore, the applied optimization technique should be able to continuously search for the best solution. Evolutionary algorithms (EAs) and other adaptive search techniques, such as simulated annealing, fulfill this requirement. However, EAs have the additional advantage that they maintain a population of solutions throughout the run rather than just a single solution. Thus, the search for new solutions can take advantage of the diversity of the population and the competition between the individuals. Each available candidate solution offers a starting point for discovering new ways to treat the problem given whatever change has occurred. Therefore, the optimization process does not have to rely on a single starting point only, and new solutions do not have to be recomputed from scratch. For reasonably small changes of the problem, some individuals in the present population are likely to be in close vicinity of optima in the next generation.

There are four main issues to consider regarding optimization in control problems:

First, computation time is a critical factor. In most cases, the optimization technique is applied to a simulator before its solutions are transferred to the real system. Realistic simulators of systems, such as industrial production control or vehicle steering control, are usually complex and evolutionary optimization may take days on state-of-the-art computers. An example is the complex simulation of the temperature field in a slab of continuously cast steel, which has to be computed to evaluate the fitness of each candidate solution for the controller (Filipič and Šarler, 1998). This limits the experimental feasibility regarding population size and number of generations. Furthermore, the time needed for the fitness evaluation of a candidate solution can be a serious limitation regarding real-time applications. In so-called direct optimal control (Fogarty et al., 1995), an EA continuously evolves the settings of the control parameters, i.e., the EA is not tuning or evolving other controllers such as a PID or a fuzzy controller. Such an online-evolution process faces the problem that the longer it takes to compute new solutions the more the fitness landscape can change in real time, which makes the problem increasingly harder. Therefore it is of critical importance to achieve an optimal balance between the number of evaluations and the required computation time as well as between the population size and the number of generations per time-step.

Second, in contrast to dynamic observation problems, in control problems there is feedback between the controller and the system. Each control action consequently affects the shape of the fitness landscape. In other words, the search for the optimal control affects the problem.

Third, control has to be robust. A typical problem in hardware control is the drift of material parameters due to machinery wear-out and sensitivity to environmental changes, such as temperature, light, and humidity (Filipič and Juričić, 1993).

Finally, the pure performance output of the system is often inappropriate to serve as a fitness evaluation, but rather the design of the fitness function itself is of critical importance for the success of the optimization (Filipič et al., 1999).

We suggest three main classes of control problems: (i) state stabilization (e.g. constant electricity supply of a power plant), (ii) system-to-system interaction (e.g. 'arms-race' with another system, such as pest control versus a crop pest), and (iii) profit maximization (e.g. market-oriented production optimization in a factory).

In this paper, we investigated control optimization strategies in profit maximization by the example of a simulated crop producing greenhouse. In this setup, we experimented with different control strategies, which were based on the evaluation of a limited amount of candidate solutions per simulated system time-step.

Our motivation for the design of the greenhouse simulator was that there are hardly any appropriate benchmark tests or test case generators for control problems until today. An exception in state stabilization is the pole-balancing problem (Karr, 1991), which is a well-known standard problem that is relatively easy to implement. Classic benchmark tests in numerical optimization, such as the Rastrigin or De Jong functions, are usually simple and static. In control problems, the demands on the optimization process are very different and performance results from experiments with simple static functions are of little use due to the issues outlined above. Also the recently suggested test case generators (TCGs) for dynamic environments (e.g. (Branke, 1999; Morrison and Jong, 1999; Grefenstette, 1999)) are insufficient to mimic control problems. Their main weakness is that they arbitrarily distort the landscape over time by moving or bouncing peaks. Not surprisingly, it remains unclear how much and in which way these landscape distortions would resemble characteristics of real-world problems. Another major limitation of these TCGs regarding control problems is that they cannot simulate the feedback between the control and the system, i.e., that the search affects the change of the landscape. Moreover, the frequency and degree in which fitness landscapes are changed over time is specified by independent parameters, whereas in real systems the speed of changes between two successive control actions is mainly affected by the time it takes to evaluate the candidate solutions. For reasonable test cases, a realistic interaction between the speed of changes and the evaluation of individuals is essential, since the speed ultimately determines the success of the optimization process, see e.g. (Ursem, 2000).

These limitations of previously suggested TCGs have been recently addressed by a study of two of the authors and other collaborators (Ursem et al., subm). The result of this study was a new test-case simulator language that supports the modeling of dynamic test case problems by simulation of real-world systems. The main idea was to create a tool that allows to model instances of time-varying fitness landscapes by the performance output of a modeled system. As an example, the control problem of a crop producing greenhouse was implemented, which we adopted and refined in this study. In contrast to the earlier paper, our main intention with the present paper was to focus on the principle issue of the design of the control strategy.

The remaining sections of the paper are structured as follows: In Section 2, we present a general design of benchmark control problems. Afterwards, in Section 3, we introduce the greenhouse model with all its components and parameters. Section 4 contains the results of our preliminary experiments with a set of control strategies for production control optimization, and finally, in Section 5, we discuss the results of this study.

2 DESIGN OF BENCHMARK CONTROL PROBLEMS

Control problems in engineering are often represented by the interaction between the controller and the controlled system (Figure 1). Here $\mathbf{x}(t)$ represents the internal state of the system at time t, $\mathbf{u}(t)$ is the control signal, and $\mathbf{y}(t)$ is the output from the system.

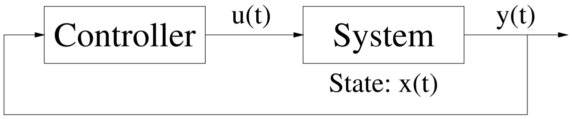


Figure 1: Controller and the system being controlled.

Further, it is often necessary to model the environment that surrounds the system if it, in addition to the controller, affects the system (see Figure 2).

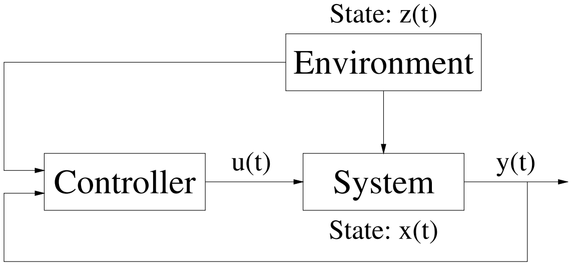


Figure 2: Model for controller, system, and environment.

The representation of a system requires to identify all variables that are either a part of the system or influence it indirectly. Further, the type and range of each variable (control, system, or environment) needs to be determined. It is not always clear where to draw the line between control, system, and environment; however, the following simple rules can be used to categorize variables into one of the three classes.

1. Control variables are variables that can be set by the controller, e.g., outlet of a valve.
2. System variables are directly affected by the control variables, but also through interactions with other variables. For instance, the level in a rainwater tank.
3. Environment variables represent components that influence the system, but are not directly affected by the control variables, e.g., amount of rainfall.

The time-varying system and environment states can often be modeled by difference equations of the form

$$x_i(t+1) = x_i(t) + \Delta x_i(t) \qquad (1)$$

where $\Delta x_i(t)$ usually depends on other variables of the controller, the system, and the environment.

2.1 TECHNICAL ASPECTS

As stated in the introduction, the application of EAs to control problems has the side-effect that the search changes the problem. Hence, the candidate solutions in a population have to be evaluated from the same starting state. Consequently, the entire state of the simulator has to be stored and restored between evaluations. After all candidate solutions within one time-step have been evaluated, one solution, usually the best, controls the system for a number of simulation steps. From this state on the process is repeated. Figure 3 illustrates an abstract scenario where four control settings are evaluated for three time-steps. The best setting is then used to control the system one time-step.

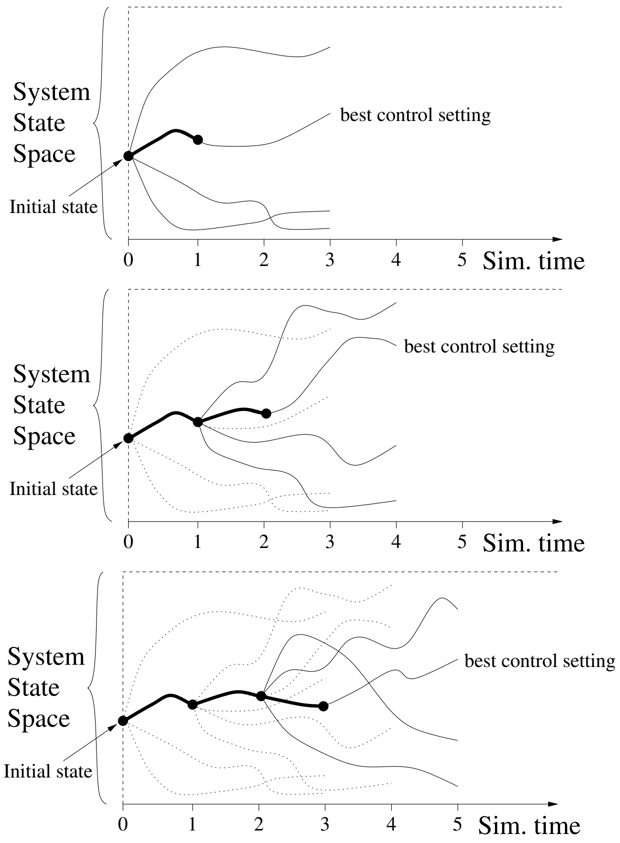


Figure 3: Example of state space exploration at simulation time $t = 0$, $t = 1$, and $t = 2$. Thin lines represent control settings exploration of the current time-step, thin dotted lines are previous explored control strategies, and thick lines are actual control as it was performed by the selected control setting.

3 THE GREENHOUSE MODEL

As a benchmark test for our study, we implemented a simple simulator for a crop producing greenhouse. The production is controlled by heating, injection of CO_2, ventilation, and optional use of artificial light. All produced crops are sold at a time-varying market price. The objective is to maximize the profit, i.e., to maximize the production while minimizing the expenses of heating, CO_2, and electricity.

The simulator consists of three types of interacting variables and can store and restore the state before the fitness evaluations. The implemented model only represents a simplified subset of the components found in real greenhouses; however, it still illustrates interesting characteristics of greenhouse control.

The model consists of the following variables:

Control variables:
u_{heat} – Heating. $u_{heat} \in [0, 1]$
u_{vent} – Ventilation. $u_{vent} \in [0, 1]$
u_{CO_2} – CO_2 injected. $u_{CO_2} \in [0, 1]$
u_{light} – Artificial light. $u_{light} \in [0, 1]$

System variables:
x_{itemp} – Indoor temperature. $x_{itemp} \in [-20, 60]$
x_{CO_2} – CO_2 level in the greenhouse. $x_{CO_2} \in [0, 5]$
x_{crop} – Amount of harvested crop. $x_{crop} \in [0, \infty]$

Environment variables:
z_{otemp} – Outdoor temperature. $z_{otemp} \in [-20, 60]$
z_{sun} – Sunlight intensity. $z_{sun} \in [0, 1]$
z_{pcrop} – Price for crops. $z_{pcrop} \in [6.75, 7.25]$
z_{pheat} – Price for heating. $z_{pheat} \in [1.25, 1.75]$
z_{pCO_2} – Price for CO_2 gas. $z_{pCO_2} \in [1.75, 2.25]$
z_{pelec} – Price for electricity. $z_{pelec} \in [0.75, 1.25]$

During the simulation, each system variable is updated using Equation 1 (constants are listed in Table 1). A step in the simulator corresponds to 15 minutes, i.e., a day consists of 96 steps.

The indoor temperature is changed by

$$\begin{aligned}\Delta x_{itemp} &= k_1 \cdot u_{heat} + k_2 \cdot z_{sun} + \\ &\quad (k_3 + k_4 \cdot u_{vent})(z_{otemp} - x_{itemp})\end{aligned}$$

where k_1 is the temperature increase due to heating, k_2 is the increase from sunlight radiation, k_3 is the minimal heat exchange with the environment, and k_4 is the exchange rate when ventilation is used.

The change in indoor CO_2 level is modeled by

$$\begin{aligned}\Delta x_{CO_2} &= -k_5 \cdot \Delta x_{crop} + k_6 \cdot u_{CO_2} + \\ &\quad (k_7 + k_8 \cdot u_{vent})(k_9 - x_{CO_2})\end{aligned}$$

where k_5 is the CO_2 consumption by the plants, k_6 is the increase due to injected CO_2, k_7 is the minimal CO_2 exchange with the environment, k_8 is the exchange by ventilation, and k_9 is the atmospheric CO_2 level.

The crop production per time-step is somewhat more complex. The actual growth per time-step is modeled as a percentage of the optimal growth, i.e., the growth under optimal conditions of temperature, light, and CO_2 level. The change in crop growth is

$$\Delta x_{crop} = k_{10} \cdot \min(G_{temp}, G_{light}, G_{CO_2})$$

where k_{10} is the maximal amount of produced crops. The min-function models that plant growth is limited by the smallest "growth-percentage". For instance, if $G_{temp} = 0.73$, $G_{light} = 0.35$, and $G_{CO_2} = 0.98$, then the current production is at 35%. The impact of these variables on growth is described by growth transfer functions. They are illustrated in Figure 4. The transfer function for temperature models an optimal growth temperature of 30 degrees Celsius with a near optimal range of 25 to 35 degrees. The intervals from -20 to 0 degrees and 45 to 60 degrees do not allow any growth. In fact, the plants die and have to be replanted if the indoor temperature is not between 0 to 45 degrees. In this case, no production is possible for 30 days ($\Delta x_{crop} = 0$). The transfer function for the CO_2-level models a saturation effect. The light transfer function maps both sunlight and artificial light to a production percentage, which is also modeled as a saturation relationship.

The profit in a time-step is calculated as follows:

$$\begin{aligned}p_{profit} &= z_{pcrop} \cdot \Delta x_{crop} - (z_{pheat} \cdot u_{heat} + \\ &\quad z_{pCO_2} \cdot u_{CO_2} + z_{pelec} \cdot u_{light})\end{aligned}$$

Real weather data were available for the environment variables z_{otemp} and z_{sun} (see Figure 5). The data represent a typical year in Denmark. The subset corresponding to March was used in the simulations. March in Denmark has days with both freezing and non-freezing temperatures. The prices were updated every 7-14 days (randomly determined) and set to a random value in the variable's interval (rounded to steps of 0.05).

Table 1: Constants for Δ-functions.

$k_1 = 0.5$	$k_2 = 0.3$	$k_3 = 0.005$	$k_4 = 0.1$
$k_5 = 0.15$	$k_6 = 0.5$	$k_7 = 0.05$	$k_8 = 1$
$k_9 = 3.0$	$k_{10} = 3.0$		

4 EXPERIMENTS AND RESULTS

The objective in the experiments was to test trade-offs between population size and number of generations using the previously described direct control strategy.

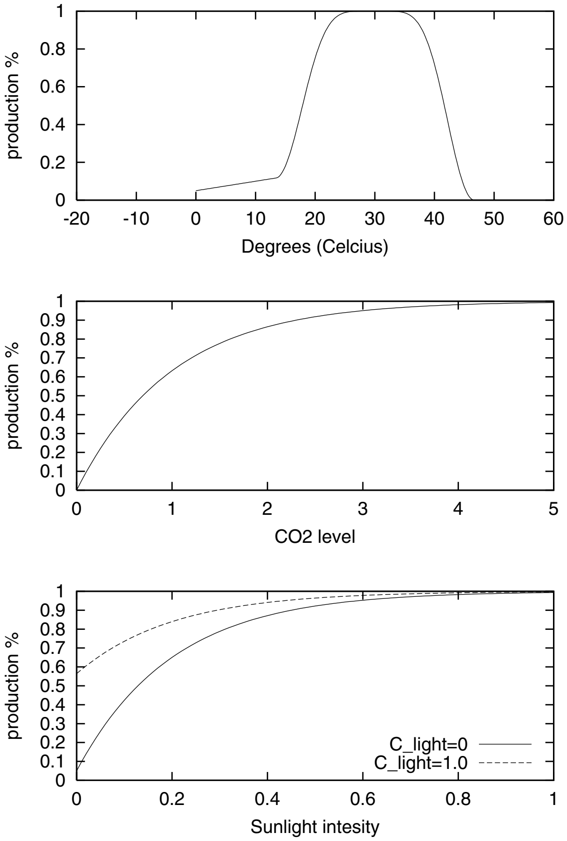


Figure 4: Growth transfer functions for Δx_{crop}. Upper graph: G_{temp}, middle graph: G_{CO_2}, and lower graph: G_{light} with and without artificial light.

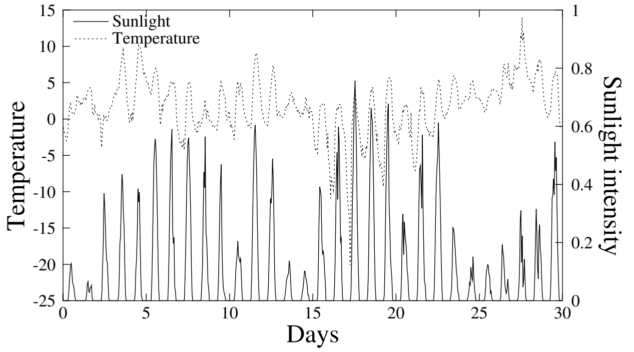


Figure 5: Temperature and sunlight intensity for March in Denmark.

We implemented a classic GA to evolve the four control settings of the greenhouse simulator. For this, we used real-valued vectors, arithmetic crossover, and Gaussian mutation to evolve the four control settings. Probability of crossover $p_c = 0.9$, probability of mutation $p_m = 0.5$, and variance $\sigma = 0.5$. The search space was discretized, such that u_{heat}, u_{vent}, and u_{CO_2} had steps of size 0.01, whereas the artificial light u_{light} had three settings 0, 0.5, and 1.

Each solution was evaluated by simulating eight steps (2 hours) using the control setting encoded in the genome. The profit achieved in each time-step was recorded and used to calculate the fitness. Two fitness calculation functions were tested, i) sum of profit

$$Fit_{sum}(I) \quad = \quad \sum_{i=1}^{8} p_{profit}[i] \tag{2}$$

and ii) weighted sum of profit

$$Fit_{wei}(I) \quad = \quad \sum_{i=1}^{8} w[i] \cdot p_{profit}[i] \tag{3}$$

where $w = [1.0, 0.875, 0.75, ..., 0.125]$ and $p_{profit}[i]$ denotes the recorded profit in the i-th measurement in the simulation. The second evaluation methods “rates” near future profit higher than profits of a more distant future.

The total number of evaluations provides a basis for comparison. To examine the trade-off between population size and number of generations, we kept the total number of evaluations constant. Two series of experiments were performed; one with 200 evaluations and one with 400. The tested trade-off settings are listed in Table 2.

Table 2: Test cases for trade-offs between population size and generations before update.

# Evals	Pop size	# Generations
200	200	1
	100	2
	50	4
	25	8
400	400	1
	200	2
	100	4
	50	8

The results from the experiments are listed in Table 3. Figure 6 shows the average daily profit for 200 evaluations (popsize=100, # generations=2) for both fitness methods.

Figure 7 illustrates two typical fitness landscapes where u_{heat} is plotted vs. u_{CO_2}. The variables u_{vent} and u_{light} were set to constant values according to the currently best individual to allow a 3D visualization of the fitness landscape.

Table 3: Mean and standard error of total profit (average of 100 runs).

		Pop size	# Gen	Profit	Std. error
200 Evaluations	Sum	200	1	39943.73	± 63.79
		100	2	40097.91	± 63.20
		50	4	40149.43	± 70.40
		25	8	40115.95	± 56.60
	Weight	200	1	39778.90	± 65.29
		100	2	39846.94	± 69.85
		50	4	39908.78	± 68.19
		25	8	39748.82	± 57.88
400 Evaluations	Sum	400	1	39912.91	± 60.25
		200	2	40203.79	± 58.29
		100	4	40028.13	± 60.55
		50	8	40068.30	± 60.86
	Weight	400	1	39706.74	± 62.06
		200	2	39778.89	± 60.19
		100	4	39881.62	± 58.54
		50	8	39790.53	± 60.86

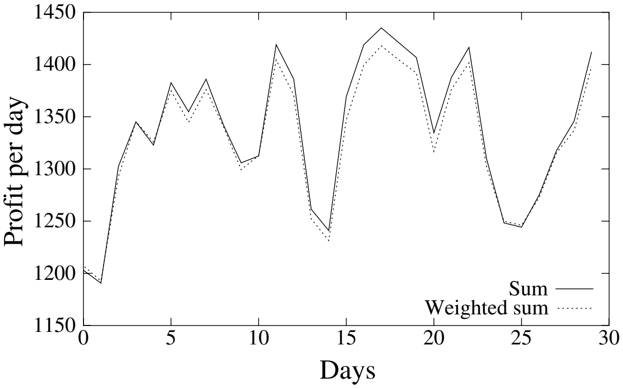


Figure 6: Average of daily profit in March with 200 evaluations (popsize=100, # generations=2) for sum and weighted sum fitness. Average of 100 runs.

5 DISCUSSION

In this paper, we investigated different strategies for control optimization by the simulation example of profit maximization in a crop producing greenhouse. One of the issues that we examined was how to design the fitness function, such that it estimates the quality of a solution in respect to its impact in the near future. The control strategy calculating the fitness as the simple sum of all profits over the look-ahead time yielded better results than the strategy using a weighted sum to give greater priority to profits in the near than the distant future. This was particularly the case when the profit was high because of beneficial weather conditions. Overall, weather conditions tightly constrained the achievable profit, i.e., cold weather with almost no sunlight required an inevitably higher investment in heating and electricity for the artificial light.

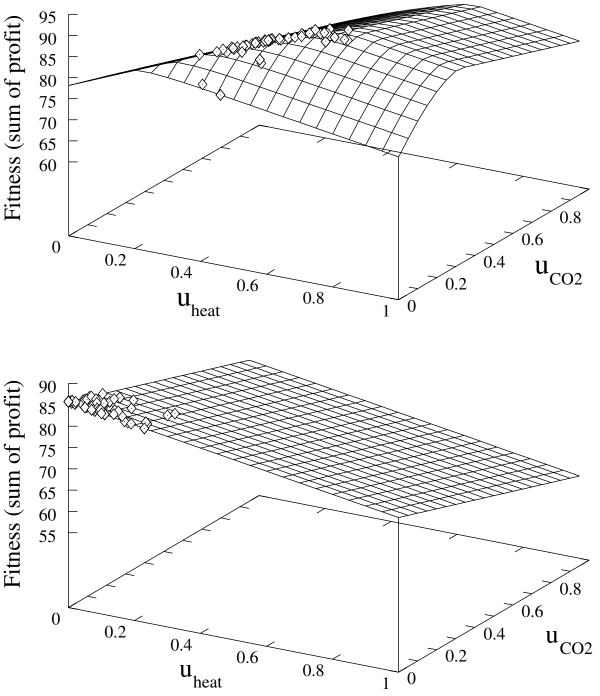


Figure 7: Two typical fitness landscapes. Plots illustrate the control variables u_{heat} versus u_{CO_2}. The variables u_{vent} and u_{light} were set to constant values according to the currently best individual to allow a 3D visualization of the fitness landscape. Diamonds represent the individuals (i.e. the genotypes) of the GA population.

Regarding the time critical problem of fitness evaluations, our results show that the performance of the investigated strategies with different investments in population size versus generations (see Table 2) were rather small. The investment strategies with more than one generation evaluation per time-step turned out to be superior compared with strategies based on only one generation and a maximum number of individuals. However, there was no conclusive ranking pattern among strategies that were evaluated more than one generation. Furthermore, doubling the number of evaluations from 200 to 400 did not yield any significant improvements.

Regarding the greenhouse benchmark test, we found that the generated fitness landscapes resembled a fundamentally different dynamic behavior than what can be simulated with previously suggested dynamic TCGs (e.g. (Branke, 1999; Morrison and Jong, 1999; Grefenstette, 1999)). This result confirms our very recent study on a new test case generator language for dynamic problems (Ursem et al., subm). The greenhouse fitness landscapes turned out to be simple tilted planes when the optimization process was far from the optimum (see Figure 7, bottom). Indeed, this result is not surprising if one considers that a controller can only set control variables within a certain limited range and the target value may only be reached after some time. For instance, to heat up a freezing cold room with a radiator, the best possible setting is to turn up the heat regulator to its maximum until the temperature approaches a pleasant level. Only at this point, the radiator control needs to be more sophisticated. In general, control problems require simple means when the current system state is far from the target state. In multiple dimensions this means that if one or more parameters are poorly controlled then the entire landscape will be tilted in these parameter dimensions. Further, fitness landscapes at more interesting system states turned out to be unimodal with characteristic ridges (see Figure 7, top) rather than multimodal, cone-shaped peaks as suggested by earlier dynamic TCGs. Characteristically for control problems, the search for optimal control affected the transformations of the fitness landscape, which is another property that cannot not be simulated by earlier dynamic TCGs. Although our current implementation of the greenhouse simulator is very simple and preliminary, the principle characteristics of the resulting landscapes reflect what we would expect from realistic dynamic landscapes in control optimization. However, so far the simulator does not pose a very challenging control optimization task. Delay factors, such as a gradual warming-up of the heating system or the rather slow diffusion of the CO_2 gas, would make the control problem harder and more realistic.

Apart from the greenhouse model as a representative for profit maximization problems, it would be important to investigate instances of the other two suggested control problem domains (state stabilization and system to system interaction) in the future. Furthermore, it would be interesting to make an elaborate performance comparison between a classic GA and more advanced techniques, such as Multinational GAs (Ursem, 1999), Religion-based EAs (Thomsen et al., 2000), or mass extinction techniques (Krink et al., 2000; Greenwood et al., 1999), which clearly outcompete classic GAs in numerical benchmark tests. The ability of Religion-based EAs and so-called SOC mass extinction EAs to maintain a much higher population diversity than the classic GA (Thomsen and Rickers, 2001) might turn out to be of particular benefit in control optimization. Moreover, it remains an open question how well other search heuristics, such as hill climbers, like next ascent, steepest ascent, and simulated annealing (Davis, 1987), would cope with control optimization compared to EAs. Also regarding the control strategy there are still open questions related to the design of the fitness function in respect to the quality evaluation of a candidate solution over time. One interesting possibility would be the implementation of self-adaptive look-ahead strategies. Clearly, the present paper is only a first and preliminary attempt to tackle these questions and further research in this area will be required.

Acknowledgements

The authors would like to thank the Danish Research Council, the Ministry of Education, Science and Sport of the Republic of Slovenia, and the Institute for Advanced Study, Berlin, for financial support.

References

[Branke, 1999] Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1875–1882, Washington D.C., USA. IEEE Press.

[Davis, 1987] Davis, L., editor (1987). *Genetic Algorithms and Simulated Annealing*. Research Notes in Artificial Intelligence. Pitman Publishing, London.

[Filipič and Juričić, 1993] Filipič, B. and Juričić, D. (1993). An interactive genetic algorithm for controller parameter optimization. In Albrecht, R. F., Reeves, C. R., and Steele, N. C., editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms ANNGA '93*, pages 458–462, Innsbruck, Austria. Springer-Verlag.

[Filipič et al., 1999] Filipič, B., Urbančič, T., and Križman, V. (1999). A combined machine learning and genetic algorithm approach to controller design. *Engineering Applications of Artificial Intelligence*, 12(4):401–409.

[Filipič and Šarler, 1998] Filipič, B. and Šarler, B. (1998). Evolving parameter settings for continuous casting of steel. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing EUFIT'98*, volume 1, pages 717–721, Aachen, Germany. Verlag Mainz.

[Fogarty et al., 1995] Fogarty, T. C., Vavak, F., and Cheng, P. (1995). Use of the genetic algorithm for load balancing of sugar beet presses. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 617–624, San Francisco, CA. Morgan Kaufmann.

[Greenwood et al., 1999] Greenwood, G. W., Fogel, G. B., and Ciobanu, M. (1999). Emphasizing extinction in evolutionary programming. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the Congress of Evolutionary Computation*, volume 1, pages 666–671, Washington D.C., USA. IEEE Press.

[Grefenstette, 1999] Grefenstette, J. J. (1999). Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 2031–2038, Washington D.C., USA. IEEE Press.

[Karr, 1991] Karr, C. L. (1991). Design of an adaptive fuzzy logic controller using a genetic algorithm. In Belew, R. and Booker, L., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 450–457, San Mateo, CA. Morgan Kaufman.

[Krink et al., 2000] Krink, T., Thomsen, R., and Rickers, P. (2000). Applying self-organised criticality to evolutionary algorithms. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature – PPSN VI*, volume 1, pages 375–384, Paris, France. Springer.

[Morrison and Jong, 1999] Morrison, R. W. and Jong, K. A. D. (1999). A test problem generator for non-stationary environments. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 2047–2053, Washington D.C., USA. IEEE Press.

[Thomsen and Rickers, 2001] Thomsen, R. and Rickers, P. (2001). Introducing spatial agent-based models and self-organised criticality to evolutionary algorithms. Master's thesis, University of Aarhus, Denmark.

[Thomsen et al., 2000] Thomsen, R., Rickers, P., and Krink, T. (2000). A religion-based spatial model for evolutionary algorithms. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature – PPSN VI*, volume 1, pages 817–826, Paris, France. Springer.

[Ursem, 1999] Ursem, R. K. (1999). Multinational evolutionary algorithms. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the Congress of Evolutionary Computation (CEC-99)*, volume 3, pages 1633–1640, Washington D.C., USA. IEEE Press.

[Ursem, 2000] Ursem, R. K. (2000). Multinational GAs: Multimodal optimization techniques in dynamic environments. In *Proceedings of the Second Genetic and Evolutionary Computation Conference (GECCO-2000)*, volume 1, pages 19–26, Las Vegas, USA. Morgan Kauffmann Publishers.

[Ursem et al., subm] Ursem, R. K., Krink, T., Jensen, M. T., and Michalewicz, Z. (subm). Analysis and modeling of control tasks in dynamic systems. *IEEE Transactions on Evolutionary Computation.*

Adaptive Evolvability via Non-Coding Segment Induced Linkage

C.-Y. Lee
Department of Mechanical Engineering
California Institute of Technology
cinyoung@caltech.edu

E.K. Antonsson
Department of Mechanical Engineering
California Insitute of Technology
erik@design.caltech.edu

Abstract

In this paper, the evolvability, or "the ability of a population to produce fitter variants than any yet existing" [1], of a genetic algorithm is adapted during runtime by evolving good linkage characteristics. This adaptive linkage is induced by a variable length non-coding segment representation. It is expected that the adaptation of linkage will improve search efficiency due to the emergence of well linked, or modular, genetic code. Results of the non-coding representation are compared with those of a standard representation on a set of Royal Road functions.

1 INTRODUCTION

While the origins of life have been dated to roughly 3.5 billion years ago, multicellular organisms did not appear until 2.5 billion years later. Approximately half a billion years after this event, or roughly 500 million years ago, a rapid proliferation in the variety and complexity of organisms (as observed from the fossil record) occurred in what is now called the Cambrian explosion. This sudden increase in evolutionary development has been partially attributed to the acquisition of hierarchial modules into the genome [2]. The presence of genetic modules that code for independent attributes allows for rapid experimentation of new body types (*e.g.*, variation of eye or limb number); since, rather than having to evolve completely new body parts, genetic modules can be reused or exploited. So, in a sense, the newer modular genetic codes can be considered to be more *evolvable* than the previous non-modular, genetic codes. Altenberg provides a more precise definition of *evolvability* as "the ability of a population to produce variants fitter than any yet existing" [1].

The concept of evolvability is central to the design of evolutionary algorithms because a minimum level of evolvability is required for convergence to any extrema. For example, recall the problem of computer program synthesis by evolutionary methods. Early approaches represented programs as strings of letters and generated new programs through random variation of the letters. This approach has little or no evolvability, since fitter variants of functional programs are found with near zero probability. In contrast, Koza's genetic programming (GP) approach represented programs as trees and used subtree crossover to generate offspring programs. Such an approach has the requisite evolvability to produce fitter variants as demonstrated by GP's many successes. It is apparent that evolvability is a characteristic governed by the highly interrelated choice of representation and search operators (since they determine how new variants will be created). Moreover, once a a suitable level of evolvability has been attained through intelligent choice of representation and search operators, it is desirable to have the population's evolvability increase during runtime to improve search efficiency.

In this paper, a novel, adaptive representation based on non-coding segments is developed to address the issue of adapting a population's evolvability in a genetic algorithm. The remainder of the paper introduces this development and is organized to: (i) review previous related work, in particular the use of non-coding segments in genetic algorithms, (ii) develop a new variable length non-coding segment representation, (iii) present results of a genetic algorithm implementing the non-coding segment representation on a set of Royal Road functions, and (iv) conclude with a summary and discussion of future work.

2 PREVIOUS WORK

A tremendous amount of work exists on the adaptation of evolvability (although, not always classified in this sense) and is too extensive to be thoroughly discussed here. A small sample of previous work is presented for illustration. A classic example of adaptive evolvability in evolutionary algorithms is the implementation of self-adaptive mutation operators in evolution strategies [3]. In this case, the mutation operator adapts its step size, or, equivalently, the radii of its search neighborhood, in accordance with the performance landscape. Conversely, messy genetic algorithms (mGAs) adapt evolvability through the use of an adaptive representation [4]. Genes in a mGA are tagged with a gene marker that allows reordering of genes, which, in turn, allows good linkages to develop within the chromosome. Linkage is defined here as the probability that two genes will remain together after crossover. A good linkage would then keep two related genes tightly linked so that they would remain together after crossover has occurred. In all GAs, adjacent genes are more tightly linked than non-adjacent genes. By allowing the adjacency of genes to change, mGAs can find better linkage characteristics. However, in contrast to nature, linkage between genes in mGAs (and canonical GAs, for that matter) are fixed and equal (*i.e.*, the probability of dirsuption between any pair of genes is constant and equal to that of any other pair with the same separation). The occurrence of linkage variation in nature has been partially attributed to the presence of non-coding segments, or 'junk' DNA, in biological genomes [5].

2.1 NON-CODING SEGMENTS

In nature, the majority of DNA in an organism is non-coding, meaning that it is not transcribed into RNA for protein synthesis. The reasons for non-coding DNA are not well known. It has been hypothesized that non-coding DNA prevent the destruction of good building blocks (such as genes) during recombination since crossover is more likely to occur in a region of non-coding DNA [5]. Furthermore, genes with little or no non-coding regions between them will have tight linkages. Thus, modular genetic structures can develop as a result of the variability in non-coding segment length. This premise of modularity, a hallmark of increased evolvability, has led to several studies on the utility of non-coding segments in genetic algorithms.

Levenick, Forrest, and Wu all implement similar non-coding segment representations in which a fixed number of non-coding bits are inserted between coding bits within the chromosome [6, 7, 8, 9]. In each work, the representation is tested on modular fitness functions whose modules, or building blocks, are known. This *a priori* knowledge allows the experimenters to place the non-coding segments between adjacent modules and observe the effectiveness of such an approach. However, these experiments do not address the issues of proper non-coding segment placement and length – or, how one might determine good linkage characteristics without *a priori* knowledge. The work developed in this paper addresses these issues by introducing variable length, non-coding segments and is discussed in the subsequent section.

The work done by Lobo on compressed introns in a linkage learning genetic algorithm has some interesting parallels to the current work [10]. In addition, it would be remiss not to mention Levenick's follow up to his previous work [11] or the numerous investigations of non-coding segments in genetic programming, which are not cited here.

3 VARIABLE LENGTH NON-CODING SEGMENTS

A new chromosome representation is developed in which non-coding segments separate adjacent genes. This representation is an extension of the work first introduced in [12] and implemented in [13]. As opposed to the bit string approach taken by Levenick, Forrest, and Wu, non-coding segments are encoded as real numbers.

The new representation is presented by example. The following 4 bit chromosome

$$1 \quad 0 \quad 1 \quad 1$$

has a non-coding segment representation given as

$$\mathit{0.1} \quad 1 \quad \mathit{0.05} \quad 0 \quad \mathit{0.5} \quad 1 \quad \mathit{0.5} \quad 1 \quad \mathit{0.1}$$

where the italicized genes are non-coding segments that indicate the width of separation between the non-italicized, coding genes. The coding genes have no length, so the total chromosome length is simply the sum of non-coding segments. For example, the above chromosome has a total length of *(0.1+0.05+0.5+0.5+0.1)=1.25.* As opposed to encoding the widths between adjacent coding genes, the non-coding segments can be made to encode the cumulative gene length up to the following coding gene. This equivalent representation is shown for the preceding chromosome as

$$\mathit{0.1} \quad 1 \quad \mathit{0.15} \quad 0 \quad \mathit{0.65} \quad 1 \quad \mathit{1.15} \quad 1 \quad \mathit{1.25}$$

Such a cumulative length representation permits a mapping of the coding genes where the location of each gene is given by the preceding non-coding segment. This representation scheme is introduced as it is more amenable to the crossover operator described below, which is a straightforward extension of canonical recombination operators.

Rather than choosing a range of gene indices over which to swap, a range of length values between zero and the total length is chosen. For example, say that the crossover range is chosen as 0.2 to 0.7. Then, the second and third genes are swapped between parent chromosomes since they are the only genes located in the range 0.2 to 0.7. Moreover, ranges such as 0.7 to 0.2 are valid due to the adopted cyclic chromosome structure. Because the crossover operator chooses the range endpoints from an uniform distribution over zero to the total length and if the non-coding segment lengths vary between genes, gene linkage will also vary between genes. This is easily observed in the above chromosome, where the first and second genes, which are separated by a length of 0.05, are more tightly linked than the third and fourth genes, which are separated by a length of 0.5.

Although this representation allows the encoding of different linkages between genes, there is no mechanism for adapting linkages, or non-coding segment length. A simple adaptation operator is introduced and described in the following. Prior to recombination, the non-coding segments of the first parent are perturbed by addition of a Gaussian random variable with zero mean and small variance, which is empirically chosen as 0.1. The hypothesis is that if the parent chromosome exhibits good linkage characteristics along with the correct coding genes, the chromosome will have a higher probabiliy of survival than chromosomes without good linkage characteristics. It is believed that these chromosomes will maintain good building blocks more often than other chromosomes, providing the needed advantage for offspring survival.

4 TESTING THE VARIABLE LENGTH NON-CODING SEGMENT REPRESENTATION

4.1 SEARCH PROBLEM

As with Forrest and Wu, a series of Royal Road functions are used to test the non-coding segment representation. The Royal Road functions are a set of functions with levelled fitness functions that have increasingly complex modules. These functions are useful in studying whether and how modules (or building blocks) are searched.

$$\begin{array}{ccccccccc}
l=0 & 1 & * & * & * & * & * & * & * \\
 & * & 1 & * & * & * & * & * & * \\
 & * & * & 1 & * & * & * & * & * \\
 & * & * & * & 1 & * & * & * & * \\
 & * & * & * & * & 1 & * & * & * \\
 & * & * & * & * & * & 1 & * & * \\
 & * & * & * & * & * & * & 1 & * \\
 & * & * & * & * & * & * & * & 1 \\
l=1 & 1 & 1 & * & * & * & * & * & * \\
 & * & * & 1 & 1 & * & * & * & * \\
 & * & * & * & * & 1 & 1 & * & * \\
 & * & * & * & * & * & * & 1 & 1 \\
l=2 & 1 & 1 & 1 & 1 & * & * & * & * \\
 & * & * & * & * & 1 & 1 & 1 & 1 \\
l=3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}$$

Figure 1: Royal Road function modules.

The Royal Road functions used here have 64 bits. The length of each module is $8 * 2^l$ where l indicates the level of the module and is between 0 and 3; hence, module lengths are 8, 16, 32, or 64. Each level has $64/(8 * 2^l) = 2^{3-l}$ modules that have no overlapping genes. Each module consists entirely of 1's and has an associated fitness value if contained within a chromosome. The total chromosome fitness is the sum of these module fitnesses. From this description, it can be seen that an efficient search should construct the low level modules before advancing to the higher level modules. Hence, observation of how and when modules of a certain level are created gives an indication of the search algorithm's ability to handle chromosomes with modular fitness functions.

Three different module fitness functions are used. The first is the *flat* fitness function where all modules have a fitness of 1, regardless of module length or level. The second uses a *power* fitness function where the module fitness depends on the level and is given by 3^{l+1}. The final fitness function is *exponential* and equates module fitness to module length.

For clarity, all fifteen modules are shown in Figure 1 where each gene in the figure is shorthand for an 8 bit string and $*$ is the "don't care" symbol.

4.2 GENETIC ALGORITHM DETAILS

The genetic algorithm implements the developed non-coding representation. Consequently, each chromosome has 129 genes, 64 of which are coding and 65 of which are non-coding. Chromosomes of the seed population are randomly initialized by choosing each coding gene as 1 or 0 with equal probability. The non-coding segments are chosen such that each coding gene lies within the range 0 to 2. Population size is either 64 or 128. A standard roulette wheel selection strategy is applied in which the best individual produces two offspring, on average, per generation and the worst individual produces no offspring.

Crossover is applied with 100% probability, with the aforementioned non-coding segment perturbations being applied prior to recombination. The offspring of the recombination are then subjected to mutation, which flips each coding gene with 2/64 probability. Termination of the genetic algorithm occurs either when a chromosome with the maximum fitness is found or when 1500 iterations have been completed.

4.3 RESULTS

Results are presented for the three different Royal Road functions implementing flat, power, or exponential fitness functions with population sizes of 64 or 128 individuals. In addition to testing the previously described GA, a standard 2 point crossover GA is tested along with a GA whose non-coding genes are fixed length, but take advantage of the known modularity (*i.e.*, non-coding segments have length 0.1 within modules and length 0.8 between modules). For notational convenience, results are referred to by a 3 symbol code. The first symbol (f, p, or e) indicates the type of fitness function. The second symbol is a number, indicating whether the results are for the variable length non-coding representation (*1*), standard 2 point crossover (*2*), or fixed length non-coding representation (*3*). The final symbol is either *64* or *128*, which denotes the population size.

Each genetic algorithm was run 300 times. The performance of the GAs are shown in Table 1. *Median* indicates the median generation at which the maximum fitness was found. *Average* is the average generation for convergence to the optimal fitness and σ is the respective standard deviation. *Miss* indicates the number of trials that did not find the optimal fitness within 1500 generations.

Note that for all fitness functions and population sizes the average performance of the non-coding representations are nearly as good or better than the standard representation. The median performance further corroborates this observation. However, all the performance distributions have large and overlapping standard deviations. This calls into question whether the non-coding representation does indeed perform better than the standard representation. In fact, by observing that the median is significantly lower than the average in every case, it is known that the distribution is heavily skewed to values below the average. The large standard deviation can then be attributed to the relatively few runs that took excessive amounts of time. Figure 2 depicts this graphically as a histogram of the number of runs per convergence time for f1-128, which is representative of all the other GA results. Thus, the general trend that non-coding representations perform more efficiently appears correct. Moreover, the variable length non-coding representation performs nearly as well or better than the fixed length non-coding representation. This indicates that the variable length non-coding representation is able to find appropriate linkage characteristics.

Table 1: Convergence Results

	Median	**Average**	σ	**Miss**
f1-128	353.5	429.8933	264.0272	1
f2-128	362.0	430.0567	265.5854	2
f3-128	356.5	426.3967	267.9843	10
f1-64	471.5	550.9100	311.1517	9
f2-64	499.0	585.4533	319.6713	15
f3-64	522.5	575.0600	314.1377	18
p1-128	385.5	450.4100	260.6045	4
p2-128	408.0	479.4133	291.4831	3
p3-128	365.0	441.8167	266.3116	6
p1-64	504.0	579.1567	317.0786	13
p2-64	501.0	573.2200	336.1498	14
p3-64	499.5	572.9800	308.7154	9
e1-128	404.0	473.8300	303.6770	10
e2-128	416.0	483.5033	286.0967	8
e3-128	392.0	468.1733	288.2384	3
e1-64	490.0	566.4633	322.9292	12
e2-64	497.5	565.4500	306.1123	10
e3-64	487.5	556.5233	303.6279	12

Plots of the average generation at which the GA first discovers a module of a given level are shown in Figures 3 through 5. These results were averaged over 150 runs; hence, the difference in discovery generation of the third level module and the convergence results listed in Table 1. Notably, the only large differences in these graphs occur in the final level, where the non-coding representations routinely match or outperform the standard representation.

The question remains of whether the non-coding representations evolve the correct linkages, or modularity. Figure 6 shows typical results of evolved non-coding

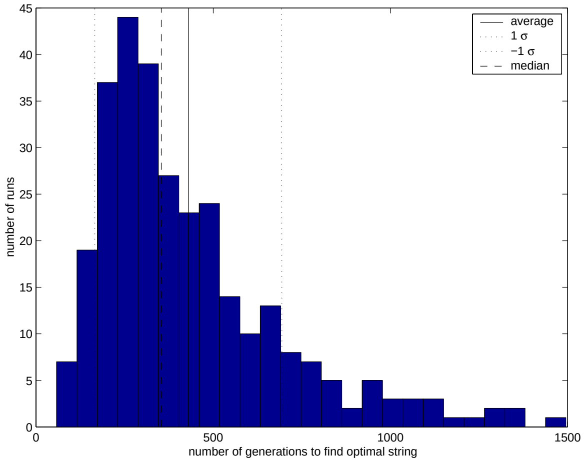


Figure 2: Distribution of number of runs per convergence time for f1-128.

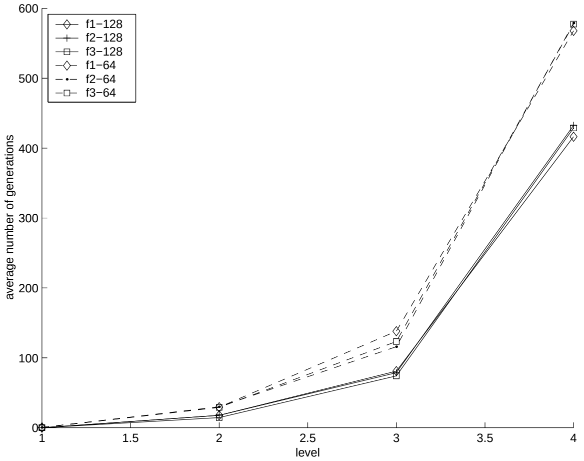


Figure 3: Average number of generations to level discovery. Flat fitness function.

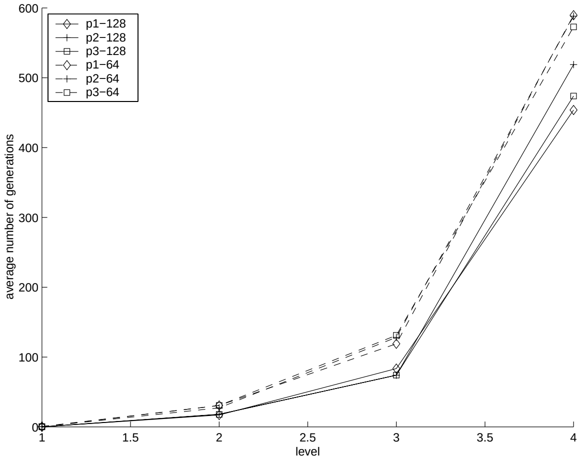


Figure 4: Average number of generations to level discovery. Power fitness function.

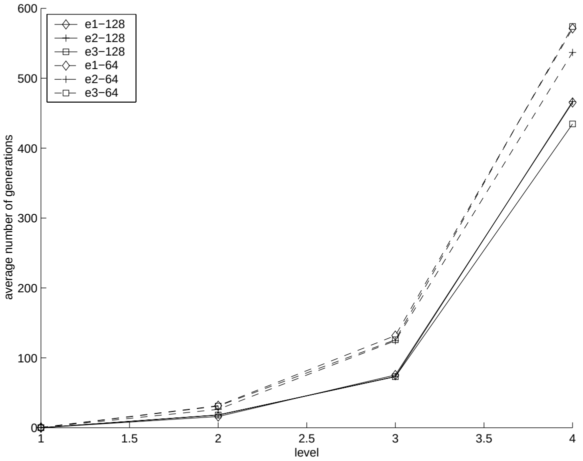


Figure 5: Average number of generations to level discovery. Exponential fitness function.

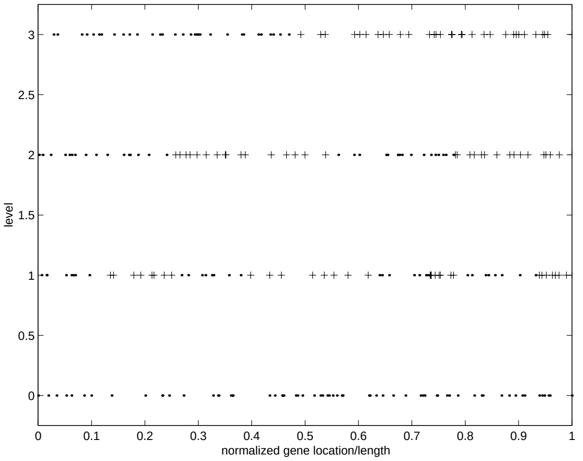


Figure 6: Normalized gene lengths at discovery of different levels for one run of p1-128.

lengths, where the normalized gene lengths are shown for the first occurrence of the module of the given level. Also, for ease of interpretation, the gene lengths are differentiated according to subsequent modules (*i.e.*, each block of '.' and '+' is a module). The zero level gene length obviously will not have the correct modularity (since it is very likely that a level 0 module occurs in the initial population), and the results reflect this. As the levels increase beyond zero, the gene lengths have some semblance to the actual Royal Road modules, with tighter clusterings within the known modules.

5 CONCLUSION

In this paper, a variable length non-coding segment representation has been developed with the goal of adapting evolvability through improved linkage characteristics. Preliminary results indicate that this representation is able to outperform a standard representation on a set of Royal Road functions. In addition, the developed non-coding representation is able to perform as well as a fixed length non-coding representation that has the known modularity built in. Furthermore, the evolved modularity (as observed by the intra-module clustering of gene locations) in the developed representation closely replicates the modularity of the known Royal Road building blocks. This implies that the developed representation can be used not only to improve the performance of standard genetic algorithms, but also to determine a rough idea of the modularity of an *a priori* unknown modular fitness function. Perhaps by observing the distribution of evolved modules in multiple runs, the actual modularity can be inferred. This is a topic of continuing research. More importantly, though, the developed representation needs to be applied to more problems to determine its effectiveness in adapting evolvability.

References

[1] L. Altenberg, "The evolution of evolvability in genetic programming," in *Advances in Genetic Programming*, K.E. Kinnear, Jr., Ed. 1994, pp. 47–74, MIT Press.

[2] D. Erwin, J. Valentine, and D. Jablonski, "The origin of animal body plans," *American Scientist*, vol. 85, no. 2, 1997.

[3] T. Back, "Self-adaptation," in *The Handbook of Evolutionary Computation.* 1997, pp. 1–23, IOP Publishing and Oxford University Press.

[4] D.E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex Systems*, vol. 3, pp. 493–530, 1989.

[5] A.S. Wu and R.K. Lindsay, "A survey of intron research in genetics," in *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature*, 1996.

[6] J.R. Levenick, "Inserting introns improves genetic algorithm success rate: Taking a cue from biology," in *Proceedings of Fourth International Conference on Genetic Algorithms*, 1991, pp. 123–127.

[7] S. Forrest and M. Mitchell, "Relative building-block fitness and the building-block hypothesis," in *Proceedings of the Foundations of Genetic Algorithms Workshop*, 1992.

[8] A.S. Wu, R.K. Lindsay, and M.D. Smith, "Studies on the effect of non-coding segments on the genetic algorithm," in *Proceedings of the Sixth IEEE International Conference on Tools with AI*, 1994.

[9] A.S. Wu and R.K. Lindsay, "Empirical studies of the genetic algorithm with non-coding segments," *Evolutionary Computation*, vol. 3, no. 2, pp. 121–147, 1995.

[10] F.G. Lobo, K. Deb, D.E. Goldberg, G.R. Harik, and L. Wang, "Compressed introns in a linkage learning genetic algorithm," in *Proceedings of the Third Annual Conference on Genetic Programming.* 1998, Morgan Kaufmann.

[11] J.R. Levenick, "Metabits: Generic endogenous crossover control," in *Proceedings of Sixth International Conference on Genetic Algorithms.* 1995, Morgan Kaufmann.

[12] C.-Y. Lee and E.K. Antonsson, "Variable length genomes for evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference.* 2000, p. 806, Morgan Kaufmann.

[13] C.-Y. Lee and E.K. Antonsson, "Dynamic partitional clustering using evolution strategies," in *Proceedings of the Third Asia Pacific Conference on Simulated Evolution and Learning*, 2000.

Using Evolutionary Algorithms to Tackle Large Scale Grouping Problems

Xiaohui Liu
Department of Information Systems
and Computing,
Brunel University,
Uxbridge, UK
Xiaohui.Liu@brunel.ac.uk
+44 (0)1895 816240

Stephen Swift
Department of Information Systems
and Computing,
Brunel University,
Uxbridge, UK
Stephen.Swift@brunel.ac.uk
+44 (0)1895 816253

Allan Tucker
Department of Information Systems
and Computing,
Brunel University,
Uxbridge, UK
Allan.Tucker@brunel.ac.uk
+44 (0)1895 816253

Abstract

Grouping problems arise in many industrial and medical applications; examples include bin packing, workshop layout design, and graph colouring. This type of problem has been successfully handled using grouping Genetic Algorithms. However in problems where there are perhaps thousands of objects to be grouped, a standard Genetic Algorithm approach can run into problems. For example with a Chromosome consisting of thousands of genes and with a sizeable population, each population becomes a burden on the resources of a computer, and crossover becomes a very time consuming operation. Within this paper, we present a method for decomposing a large number of objects into mutually exclusive subsets where within-group dependencies are high and between-group dependencies are low. The method uses an Evolutionary Algorithm approach but where the whole population is a solution to the grouping problem rather than considering many candidate solutions. This reduces the resource overheads during computer implementation and the results are promising when compared with a Hill Climbing and a clustering based approach applied to simulated email log file data.

1 INTRODUCTION

There are many practical applications involving the partition of a set of objects into a number of mutually exclusive subsets, for example [Venugopal92] looks into the grouping problem when applied to manufacturing cell formation. The objective is to optimise a *metric* defined over the set of all valid subsets, and the term *grouping* has been often used to refer to this type of problems. Examples of typical grouping applications include bin packing, workshop layout design, and graph colouring. Much research has been done on the grouping problem in different fields, and it was established that many, if not all grouping problems, are NP-hard [Garey79]. Any algorithm that is guaranteed to find the global optimum will therefore run in exponential time to the size of problem space, and a heuristic or approximate procedure is normally required to cope with most real-world problems. A variety of techniques have been proposed to develop this procedure including traditional clustering algorithms, Hill Climbing and Evolutionary Algorithms. These techniques utilise a *metric* that takes relationships or dependencies between objects into account, and partitions them into a number of mutually exclusive subsets.

1.1 GROUPING PROBLEMS

This type of problem has been successfully handled using a special type of Genetic Algorithm [Holland75, Michalewicz98] called a Grouping Genetic Algorithm [Falkenauer98]. However in problems where there are perhaps thousands of objects to be grouped, a standard Genetic Algorithm approach can run into problems. For example with a Chromosome consisting of thousands of genes and with a sizeable population (needed for the Grouping Genetic Algorithm to perform efficiently), each population becomes a burden on the resources of a computer, and crossover becomes a very time consuming operation. Within this paper, we present a method for decomposing a large number of objects into mutually exclusive subsets where within-group dependencies are high and between-group dependencies are low. The method uses an Evolutionary Algorithm approach but where the whole population is a solution to the grouping problem rather than considering many candidate solutions and thus reducing the resource overheads during computer implementation. The results are compared with a Hill Climbing algorithm and a clustering based approach applied to simulated email log file data.

We are in contact with a large commercial company and have been made aware of the challenging problem of optimally allocating users to electronic mail servers. This paper documents a feasibility study which makes use of simulated email data based upon our experiences with a number of smaller email systems. This simulated data represents the frequencies of messages being sent amongst users in such a corporate email network. Here the aim is to find an arrangement of users (objects) on a number of servers (groups) in order to minimise network traffic. The long-term goal is to extend this work to a substantial real world email environment.

1.2 PREVIOUS WORK

In [Tucker01], various heuristic and evolutionary methods were compared for grouping Multivariate Time Series (MTS) variables in order to break up a high dimensional MTS into several, lower dimensional MTS. In [Swift01], the most successful of these methods, Falkenauer's Grouping Genetic Algorithm [Falkenauer98], was applied to visual field and chemical process datasets with up to seventy-six variables. However, if the dataset to be decomposed contains *hundreds* or *thousands* of variables, the burden upon computer resources would make the search infeasible.

2 METHODOLOGY

This section discusses the methodology we use to tackle the problem of large scale grouping problems.

2.1 OVERVIEW

It is assumed that there exists a square matrix, say F, containing positive integers f_{ij} representing the strength of a relationship between object i, indexed by the row, to another object j, indexed by the column (see Figure 1). For example, these could represent the number of messages sent between two users on an email system.

$$F = \begin{array}{cc} & \text{To Object} \\ \text{From Object} & \begin{pmatrix} f_{11} & f_{21} & f_{31} & & f_{n1} \\ f_{12} & f_{22} & f_{32} & \cdots\cdots & f_{n2} \\ f_{13} & f_{23} & f_{33} & & f_{n3} \\ & \vdots & & \ddots & \\ f_{1n} & f_{2n} & f_{3n} & & f_{nn} \end{pmatrix} \end{array}$$

Figure 1. The Frequency Table used to Score Candidate User Arrangements.

This relationship matrix can then be used to assign each object to the required groups. The number of groups can either be some fixed number or within a predefined interval. Section 3.1 details the simulated dataset this method can be used to solve, and will detail the groups and the nature of the matrix F.

The general outline of the methodology is as follows: given the relationship matrix F and the number or range of the groups, an algorithm is applied that iteratively changes some random starting representation of the groups. This is done by applying a metric to judge the worth of the groups using the matrix F. The intention is that each iteration should improve the value of the groups until the arrangement is judged worthwhile.

2.2 REPRESENTATION

In order to represent candidate arrangements, a population of variable length lists was used to represent the allocation of an object to a group. For example, if these objects are to be partitioned into five groups then there are five lists representing each group containing the ID of each object allocated to that group; this is illustrated in Figure 2.

Note that k_i is the number of objects in group i and that the sum of each k_i is equal to the number of objects being grouped. Note also that it is trivial problem to extend this representation so that the number of groups falls within some range, for example between 7 and 21 groups.

$$\begin{bmatrix} \{object_{11}, object_{12}, \cdots, object_{1k_1}\} \\ \{object_{21}, object_{22}, \cdots, object_{2k_2}\} \\ \{object_{31}, object_{32}, \cdots, object_{3k_3}\} \\ \{object_{41}, object_{42}, \cdots, object_{4k_4}\} \\ \{object_{51}, object_{52}, \cdots, object_{5k_5}\} \end{bmatrix}$$

Figure 2. The Representation for Five Groups

2.3 FITNESS

The metric that we now describe makes use of the relationship matrix by considering the strength between two objects irrespective of direction. This means that only $n(n+1)/2$ elements need to be stored rather than the full table, where n is the number of objects. The evaluation metric, which we define below, is used to group objects together where they have strong mutual relationship and to separate them into different groups where the relationship is low.

The metric works as follows: each group is scored on how well the objects within it relate with each other. The metric is essentially the summation of a sub-metric evaluated on each group. The sub-metric works by creating all of the unique pairings between each of the objects in a group. It subtracts one if a pair does not have a relationship; otherwise the sub-metric is incremented by the amount of messages sent between the pair. This value is modified for each pair (for each group).

Let n be the number of objects and $\{x_1..x_n\}$ be each object to be partitioned. Let G be the list of servers and $m = |G|$ (the number of groups). Let g_i be the ith member of the list G where $1 \le i \le m$ and let $k_i = |g_i|$. The notation g_{ij} refers to the jth element of the ith list of G. G is restricted such that $\bigcup_{i=1}^{m} g_i = \{x_1..x_n\}$ and $g_u \cap g_v = \phi, \forall u \ne v$ where $k_i \ge 1$. Therefore $\sum_{i=1}^{m} k_i = n$. It is clear to see that in all cases $m \le n$. The *evaluation metric* for any fixed list G, $EM(G)$, is defined as follows:

$$EM(G) = \sum_{i=1}^{m} h(g_i) \tag{1}$$

$$h(g_i) = \begin{cases} \sum\limits_{\substack{\forall a,b \\ a \ne b \\ 1 \le a < b \le k_i}} L(g_{ia}, g_{ib}) & ,\mathrm{k_i} > 1 \\ 0 & ,\text{otherwise} \end{cases} \tag{2}$$

$$L(g_{ia}, g_{ib}) = \begin{cases} f_{ab} + f_{ba} & ,\text{if } f_{ab} + f_{ba} > 0 \\ -1 & ,\text{otherwise} \end{cases} \tag{3}$$

A similar metric has been analysed and utilised successfully in [Tucker01] to group MTS variables (the objects) based on cross correlation (the relationship matrix F). This paper also contains proofs that the metric behaves in a way that is appropriate to the problem.

2.4 THE ALGORITHMS

For the experiments, a simple Hill Climb operator, several variants of an Evolutionary Algorithm and a clustering method were used. These are described in the remainder of this section.

2.4.1 The General Algorithm

Both the Hill Climb and the Evolutionary Algorithm make use of a general algorithm. This algorithm is an iterative process whereby an operator is applied repeatedly to try and improve the fitness of the current set of groups. It takes as input the frequency matrix, F, the maximum number of function calls, *MaxCalls*, the number of object, n, and the number of groups, m:

```
Input:  F, MaxCalls, n, m
        Randomly assign the n objects
        to m groups
        i=0
        Do
           Apply Operator to groups
           i=i+1
        While (i<MaxCalls)
Output: m groups of objects
```

2.4.2 The Hill Climb

The Hill Climb simply moves one object at random from a randomly selected group and places it into another randomly selected group. The fitness of the two groups are compared before and after the operation and if an improvement is found, the amended groups replace the original.

```
Input:   F (the frequency matrix)
1        Randomly select 2 parent
         groups: p1, p2
2        Clone p1, p2 to generate 2 new
         children groups: c1, c2
3        Randomly select an object in
         group c1
4        Move the object into c2
5        Calculate fitness of c1, c2
         using equation 1 and F
6        If children's total fitness >
         parent's total fitness Then
7          Replace parents with
         children
8        Else
9          Delete children
10       End If
Output:  p1, p2 or c1, c2 according to
         combined fitness (step 6)
```

2.4.3 The Evolutionary Algorithms

All variations of these algorithms use Uniform Crossover to randomly select two parents and generate two new empty children groups, one for each parent. Each parent then assigns each object within it to its own child group with the probability based on the value of the current crossover rate, w_i. Otherwise, the object is assigned to the other child.

```
Input:   F, i (the ith object)
         Randomly select 2 parent
         groups: p1, p2
         Generate 2 new Empty children
         groups: c1, c2
         For each parent
           For each object within
           the parent
             Place object into the
             corresponding child (i.e.
             c1 for p1) with a
             probability equal to the
             current Crossover Rate
             parameter, wi, otherwise
             place the object in the
             other child
           End For
         End For
```

```
8         Calculate fitness of $c_1$, $c_2$
          using equation 1 and $F$
9         If children's total fitness >
          parent's total fitness Then
10          Replace parents with
            children
11        Else
12          Delete children
13        End If
Output:   $p_1$, $p_2$ or $c_1$, $c_2$ according to
          combined fitness (step 8)
```

Note that step 8 will allow for the event where one child group may have a lower fitness than its parent but still be chosen because the total child fitness is greater.

The effect of Crossover Rate on efficiency was investigated. Experiments were carried out which kept this value fixed and also allowed it to vary depending upon the number of function calls that had been applied so far. Two versions of this variable Crossover Rate were used and calculated as in equations 4 and 5.

$$w_i = max - \frac{(max - min) \times i}{MaxCalls} \tag{4}$$

$$w_i = min + \frac{(max - min) \times i}{MaxCalls} \tag{5}$$

where *min* is the lower limit for the Crossover Rate, *max* is the upper limit, and *i* denotes the number of function calls so far.

When the algorithm makes use of equation 4, we will refer to the variant as "Max-Min" since the rate starts off equal to *max* when function calls are equal to zero and decreases linearly until it reaches *min* when function calls equal *MaxCalls*. We will refer to "Min-Max" where equation 5 is used (the opposite is true) and when the rate is held constant, we will refer to the variant as "Fixed".

2.4.4 Partitioning Around Medoids

Standard clustering methods [Jain99] can be used to arrange *n* records into *m* groups. There are many such techniques such as K-Means clustering and hierarchical methods. However the difference between these techniques and the problem being solved in this paper is that only the distances between the *n* objects are available. However one clustering method that can work on such distance matrices is a method called *Partitioning Around Medoids* (PAM) [Kaufman87, Struyf97]. PAM will be used in this paper as a comparison to the methods presented. PAM works by firstly selecting *m* out of *n* total objects that are the closest (according to the distance matrix) to the remaining (*n-m*) objects. The fitness of these medoids is calculated by placing the remaining (*n-m*) objects in a group according to the nearest medoid and summing up all of the distances of the group members from this medoid.

These *m* selected objects are the *initial medoids*. A *Swapping* procedure is then applied until there is no improvement in fitness. *Swapping* involves generating all of the possible medoid and non-medoid pairs, evaluating the fitness of each pair, and then performing the swap that improves fitness the most. The algorithm is documented below.

```
Input:    $F$, $Iterations$, $n$, $m$
          Construct $m$ Initial Medoids,
          $d_j$ that minimise $\sum_{i=1}^{n} f_{id_j}$
          For $i$ = 1 to $Iterations$
            For all object pairs, ($i$,$j$),
            where $i$ is a medoid and $j$ is
            not a medoid
              Perform the $i$, $j$ swap
              which decreases the
              fitness the most
            End For
          End For
          Allocate each medoid to a new
          group
          Allocate the non-medoids to
          their nearest medoid
Output:   $m$ groups
```

A variation on this algorithm will also be looked at where any improvement during the *Swapping* phase will be judged according to the metric defined in equation 1. We will call this method PAM-M (Metric).

3 EXPERIMENTS

Experiments were carried out to compare the efficiency of the different algorithms. This involved running several repeated experiments over the Hill Climb and the Evolutionary Algorithms (with the differing Crossover Rates). The average learning curve for each of these experiments was then calculated. PAM and PAM-M will also be included in the comparison and each is applied only once since they are deterministic. All experiments have been tested using synthetic data which is motivated by a class of real world problem and is described in the next section.

3.1 THE TEST DATASET

Large companies with multiple offices and several thousand staff have complex network and server infrastructures. In order to keep maintenance and upgrade costs to a minimum these structures need to be carefully designed and implemented. They then need to be monitored to assess areas for optimisation. A critical part of every corporate IT infrastructure is the internal email environment. Often, these mail networks are distributed across multiple sites with each site hosting multiple servers, and each server hosting several hundred users who, in turn, may themselves be distributed across several smaller satellite offices. It is suggested that a more rigorous process should be adopted to determine the configuration of users and servers. Mail systems do provide data on mail routing which can then be analysed by standard mail administration tools. Usually this happens within the mail product itself, to trace the path of an individual message or to check out all incoming or outgoing mail for an individual. However, this functionality falls far short of the level of data analysis needed to establish efficient groupings.

The dataset that is the focus of this paper is a set of simulated electronic mail message frequencies from a single office consisted of 250 people across 5 mail servers. The frequency matrix *F* described in section 2.1 indicates how many messages were sent between two people, i.e. element f_{ij} represents how many messages were sent from person *i* to person *j*. Simulated data has been used initially to demonstrate that the method is sound; the next step (as will be discussed in section 5) will be to extend this work to real data. Our experiences with real data have shown that the frequency matrix *F* is usually around 95% sparse (i.e. 95% of the cells are zero), with a mean size of 5.0 messages and a standard deviation of 15.0 for the non-zero elements.

The objective is to minimise network traffic (generated by messages between users on different servers) by grouping users who communicate frequently onto the same server. The value of the fitness function reflects this requirement by rewarding groups where there is a lot of communication between members, and penalising groups where there is no communication.

3.2 RESULTS

This section documents the experiments involved in comparing the performance of the six methods (three Evolutionary, one Hill Climb and two PAM-based) on the 250-user synthetic dataset. Figure 3 shows the six methods in terms of their fitnesses against function calls. However, the two PAM methods appear as straight lines representing the final fitness because one iteration does not correspond directly with a function call. The four stochastic methods are averaged over ten experiments to give an indication of their general performance. Table 1 displays the fitness of the discovered groups for all methods investigated, again the stochastic methods are averaged.

The stochastic methods are allowed to run for 300 000 function calls, whereas the clustering methods are run for 100 iterations. The *max* and *min* parameters for the Evolutionary Algorithms were found to be most efficient when *max* = 0.995 and *min* = 0.95. This is discussed further in section 4.3.

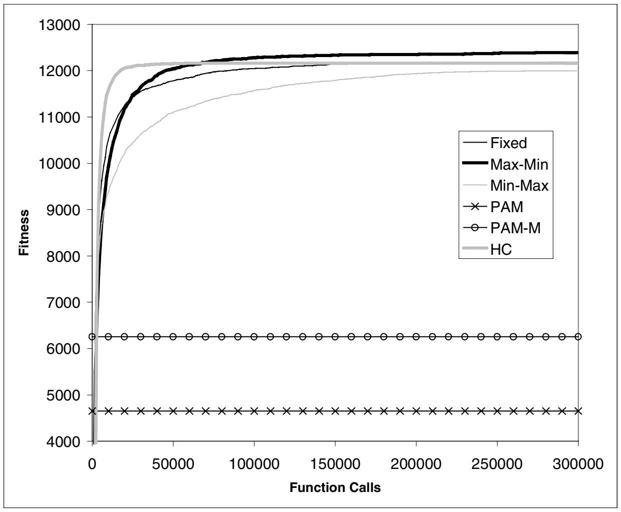


Figure 3. The Fitness of the Six Methods Plotted against Function Calls

As can be clearly seen, the Hill Climb appears very efficient within the first few thousand function calls. However as the algorithms progress, the Evolutionary Algorithms catch up. In fact the Max-Min variant soon overtakes the Hill Climb and converges to a much higher fitness. The Fixed Uniform Crossover variant is slower to converge than the Hill Climb but after approximately 170 000 function calls the two curves meet and remain virtually the same. The worst performance of the stochastic algorithms is obviously the Min-Max variant which performs poorly throughout.

Table 1. The Final Fitness of the Discovered Groups for Each Algorithm

Algorithm	Final Fitness
Evolutionary Algorithm (Fixed Uniform Crossover)	12183.7
Evolutionary Algorithm (Max-Min Crossover)	12392.0
Evolutionary Algorithm (Min-Max Crossover)	11996.6
Hill Climb	12162.0
PAM	4652
PAM-M	6253

Within table 1, the first four methods have a fractional fitness because they have been averaged over ten experiments. It is evident that the final groupings for the PAM methods compare very poorly to the other methods, particularly the standard PAM which does not make use of our metric.

4 DISCUSSION

This section considers the results that have been observed within this paper.

4.1 PAM AND PAM-M

The standard PAM method revolves around finding a *medoid* that can be considered to be the best "centre" of a group. Objects that are not medoids are placed into groups according to their closest medoid. Unfortunately, in the email application being considered, a group's suitability is judged not on how closely related a member is to a central point but how closely related each member is with its co-members. Because the email data is not the same as a distance matrix, as used by most clustering methods, a situation could arise where object A being close to object B and object B being close to object C does not necessarily imply that A is close to C. In fact, with email datasets, the opposite could easily be true.

PAM-M performs relatively better than the standard PAM method (with a 34% increase in fitness). This is most likely due to using the metric in equation 1 that takes into account all possible relationships within a group during the *Swap* phase.

4.2 HILL CLIMB

The Hill Climb makes use of small alterations within the grouping arrangement. These appear to be very effective in rapidly homing in on a good approximate allocation of objects. However, as can be seen in the experimental results as the function calls increase, the fitness grinds to a halt and does not improve any further. It was observed that the point at which this occurred varied a great deal from experiment to experiment, sometimes reaching a very good fitness and sometimes levelling out at a very early stage. This is likely to be due to the problem of encountering local maxima in the search space where the only way to improve the current grouping is by performing two or more alterations concurrently, before checking for an improvement in fitness.

4.3 THE EVOLUTIONARY ALGORITHMS

The best values for the *max* and *min* parameters were found to be higher than expected at first (*min* being 0.95 and *max* being 0.995). However, this makes sense in that as the number of objects to be grouped increases (the size of *n*) the less likely it will be that closely related objects within a group will be kept together.

The three algorithms varied a great deal in performance. The worst is the Min-Max method where the crossover rate varied from the lower bound to the higher bound. This was unexpected as it was originally thought that larger and clumsier recombinations would be useful at the beginning to rapidly locate favourable sub-groups. This would be followed by smaller changes as the experiment progressed in order to fine-tune the discovered groups. In fact, it was found that the opposite was true and the Max-Min variant performed far more favourably.

A reason for the high efficiency of the Max-Min method could be due to the early formation of sub-groups through small precise changes, behaving like schema within a standard Genetic Algorithm [Goldberg91]. As the Crossover Rate decreases, it is more likely that these sub-groups will be moved around as whole units since they are recombined with other sub-groups into complete groups which reflect the data more closely.

5 FUTURE WORK

The work presented in this paper represents the first attempt at a large grouping problem. The 250 simulated users represent a small sized office, in fact real world data often consists of tens of offices, each with thousands of users. The next stage will be to apply this work to a single office (a grouping problem involving thousands of objects) and then many offices (a grouping problem involving tens of thousands objects). Additionally the number of servers, fixed at five within this paper, should be made variable, which will require a modification to the Evolutionary Algorithm, involving perhaps the incorporation of another operator to split a group into two smaller groups. Limits will also need to be imposed on the maximum and minimum members of a group, reflecting the practical constraints that exist on the server hardware. For example, if a server has too many users allocated to it, it may function abnormally slow, and if there are too few users allocated to a server, then that server is under used and therefore not cost effective.

Focusing on the Evolutionary Algorithm itself, one of the more interesting parts of the work has been the selection of the Crossover Rate, w_i. It is known through the many experiments carried out during this project that the weights can be dependent on the number of objects, *n*. Also it is thought that the weights should be individualised for each server rather than being the same. One approach that will be looked into is whether the weights can be learnt during the execution of the algorithm, and adapt to how a particular groups fitness has been improving. Such an approach has been implemented successfully in many Evolutionary Programming implementations [Bäck93], so perhaps a technique such as self-adapting parameters [Bäck96] could be modified appropriately.

6 CONCLUSIONS

In this paper we have discussed the problems inherent in grouping large numbers of objects where the relationship/distance matrix does not behave in the manner typical of conventional clustering problems. The aim of this paper has been to demonstrate that a minimalist Evolutionary Algorithm (where the population is the solution) is best suited to this family of grouping problems, especially when the number of objects is large.

The results clearly show that a clustering based method falls down due to relying upon the notion of a centre point to a group which may not even exist. Conversely, we have demonstrated that Hill Climbing suffers from the classic problem of local maxima on such large problems. The Evolutionary Algorithms suffer from neither of these setbacks and it was discovered that the best variant made use of a variable crossover rate, contrary to what was expected.

These promising results lead us to believe that the approach adopted within this paper will extend to a class of high-dimensional real world grouping problems where other traditional methods will not be appropriate. Examples include grouping email users on servers, clustering related genes using DNA micro-array data, and grouping multivariate time series variables as in industrial process and medical domains.

Acknowledgements

We are grateful to Steve Counsell for his advice and assistance. This research is partially funded by the Engineering and Physical Sciences Research Council, UK (GR/M94120).

References

[Bäck93] T. Bäck and H.-P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization", Evolutionary Computation, 1, 1, pp. 1-23, 1993

[Bäck1996] T. Bäck, "Evolutionary Algorithms in Theory and Practice", Oxford University Press, 1996

[Falkenauer98] E. Falkenauer, "Genetic Algorithms and Grouping Problems", Wiley, 1998

[Garey79] M. Garey and D. Johnson, "Computers and Intractability - A Guide to the Theory of NP-Completeness", W.H. Freeman, San Francisco, 1979

[Goldberg91] D. E. Goldberg and K. Deb, "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms", in G. J. E. Rawlins (ed.), Foundations of Genetic Algorithms, (1), Morgan Kaufmann, pp. 69-93, 1991

[Holland75] J. H. Holland, "Adaptation in Natural and Artificial Systems", Ann Arbor, MI: The University of Michigan Press, 1975

[Jain99] A. K. Jain, M. N. Murty and P. J. Flynn, "Data Clustering: A Review", ACM Computing Surveys, 31, 3, pp. 264-323, 1999

[Kaufman87] L. Kaufman and P. J. Rousseeuw, "Clustering by means of Medoids", in Y. Dodge (ed.), Statistical Data Analysis based on the L_1-Norm, North-Holland, Amsterdam, pp. 405-416, 1987

[Michalewicz98] Z. Michalewicz and D. B. Fogel, "How To Solve It: Modern Heuristics", Springer, 1998

[Struyf97] A. Struyf, M. Hubert and P. J. Rousseeuw, "Integrating robust clustering techniques in S-PLUS", Computational Statistics and Data Analysis, 26, pp. 17-37, 1997.

[Swift01] S. Swift, A. Tucker, N. Martin and X. Liu, "Grouping Multivariate Time Series Variables: Applications to Chemical Process and Visual Field Data", To appear in: Knowledge Based Systems Journal, Elsevier, 2001

[Tucker01] A. Tucker, S. Swift and X. Liu, "Variable Grouping in Multivariate Time Series via Correlation", To appear in: IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, 2001

[Venugopal92] V. Venugopal and T. Narendran, "Cell formation in manufacturing systems through simulated annealing: An experimental evaluation", European Journal of Operational Research, 63, pp. 409-422, 1992

Knowledge-Independent Data Mining With Fine-Grained Parallel Evolutionary Algorithms

Xavier Llorà and Josep M. Garrell
Enginyeria i Arquitectura La Salle, Universitat Ramon Llull
Psg. Bonanova 8, 08022-Barcelona, Catalonia, Spain
{xevil,josepmg}@salleURL.edu

Abstract

This paper illustrates the application of evolutionary algorithms (EA) to data mining problems. The objectives are to demonstrate that EA can provide a competitive general purpose data mining scheme for classification tasks without constraining the knowledge representation, and that it can be achieved reducing the amount of time required using the inherent parallel processing nature of EA. Experiments were performed with GALE, a fine-grained parallel evolutionary algorithm, on several artificial, public domain and private datasets. The empirical results suggest that EA are competitive and robust data mining schemes that scale up better than non-evolutionary well-known schemes.

1 INTRODUCTION

Data mining with EA has to work with two main issues. The first one deals with the knowledge representation used by the algorithm for data mining. There is a wide range of choices. EA can evolve if-then rules, s-expressions, grammars or decision trees. Unfortunately, EA are usually tailored for the given dataset to be mined and the knowledge representation evolved. The second issue is common to all knowledge discovery algorithms: how the algorithm scales up with respect to the size of the search space being explored (time complexity).

Some of the most representative data mining tasks are classification, clustering or dependence modeling, among others. This paper focuses its efforts on classification problems. Classification schemes using EA like GABL [De Jong and Spears, 1991] or XCS [Wilson, 1995] are well known. They are all approaches that use Genetic Algorithms to mine datasets evolving sets of descriptive rules. Genetic Programming [Koza, 1992] has also been used to evolve s-classifiers, induction trees or classifiers based on if-then rules. All these approaches use different knowledge representations to solve the data mining problem. In fact, it is difficult to obtain a robust knowledge independent scheme for all sort of datasets. On the other hand, all of them usually share time constrains when they are used for data mining because they are designed for serial processing, and mining datasets becomes time-consuming. Parallel processing can reduce the amount of time needed to mine datasets [Freitas and Lavinston, 1998].

This paper presents GALE, a fine-grained parallel EA for data mining. It overcomes the two points discussed above. The classification scheme presented is independent of the knowledge representation evolved. This means that it is a general purpose classification scheme designed for evolving whatever is needed, sets of if-then rules, grammars or decision trees, for instance, not being tailored for any predefined knowledge representation or dataset. The second point is that it exploits parallel processing and spatial relations among individuals, reducing the amount of time required for mining the dataset. Section 2 discusses some related work. Section 3 describes GALE in detail. Section 4 presents the experimental evaluation on twelve different datasets comparing GALE to well-known classifier schemes like IB1 [Aha and Kibler, 1991], IBk [Aha and Kibler, 1991], C4.5 [Quinlan, 1993], PART [Frank and Witten, 1986], Naive Bayes [John and Langley, 1995], and SMO [Platt, 1998]. We also compare the results of GALE with other EA. Finally, section 5 summarizes our findings, discussing some conclusions and further work.

2 RELATED WORK

Recently there is a growing interest for data mining using evolutionary computation. EA have mined datasets looking for patterns indicative of time series [Povinelli, 2000], oblique datasets for complete and accurate rule sets [Wilson, 2000] and decision trees [Cantú-Paz and Kamath, 2000], or datasets for dependence modeling [Flockhart, 1995, Araujo et al., 1999]. Although they are all data mining tasks, the output knowledge is different in each work, being often tailored for the given problem. This usually leads to highly specific algorithms. Mining real-world datasets means working with attributes that are categorical, numeric, textual, etc. Thus, the point is that the data mining scheme should not constrain the knowledge representation. For instance, constraining the evolved knowledge to work only with categorical attributes narrows the scope of application of the data mining scheme.

Data mining with evolutionary computation tends to be time-consuming when mining large datasets. Parallel Genetic Algorithms (PGA) have been used to overcome the genetic search versus real time problems [Pettey et al., 1987]. There are two different ways to classify PGA, among others. The first one uses the population as reference (*population parallelism*), while the second one classifies PGA measuring the degree of parallelism on fitness computation (*fitness parallelism*). *Population parallelism* can be roughly divided in two classes: coarse-grained [Pettey et al., 1987] and fine-grained [Manderick and Spiessens, 1989] parallel algorithms. The coarse-grained approach (islands model) runs multiple populations in parallel. Among other things, when using a coarse-grained PGA one must decide the number and size of the populations (demes), the rate of migration (how often populations exchange individuals) and the destination of this migrants [Cantú-Paz, 1999]. Instead, the individuals of the population of a fine-grained PGA (cellular or diffusion model [Whitley, 1993]) are usually placed on a planar grid, where selection and crossover are restricted to small neighborhoods, exploiting the massive parallel processing available. *Fitness parallelism* [Araujo et al., 2000] divides PGA in control or application parallelism and fitness computation parallelism. The first one exploits parallelism in the application of genetic operators (such as selection, crossover and mutation), while the second one exploits inter-individual parallelism (parallelizing the fitness computation of an individual). The last one just makes sense on data mining problems where large datasets must be explored, although they might maintain serial bottlenecks that bound the speedup they can achieve (Amdahl's law [Hwang, 1993]).

PGA have reduced the amount of time spent by EA to solve all kinds of problems. They have been applied to function optimization [Spiessens and Manderick, 1991] [Gordon and Whitley, 1993], combinatorial optimization [Mühlenbein, 1989], dependence modeling [Flockhart, 1995, Araujo et al., 2000], or classifier systems [Robertson, 1987].

3 GALE

Genetic and Artificial Life Environment (GALE) is a classifier scheme based on fine-grained parallel genetic algorithms. GALE was firstly introduced in [Llorà and Garrell, 2000a], being designed for solving classification tasks. This section describes GALE focusing on the parallel evolutionary model, and how it can evolve different knowledge representations for the classification task.

The knowledge representation used for solving a classification problem can bound the classification performance and the understanding of the output knowledge obtained along the learning process [Witten and Frank, 2000]. For instance, induction trees can mine orthogonal datasets, but they have to be modified to mine oblique data. These changes in knowledge representation influence the classification performance, as well as the learning algorithm used. Evolutionary algorithms can overcome knowledge dependence providing an unified classification scheme for data mining.

3.1 CLASSIFIER SCHEME FOR DATA MINING

GALE uses a 2D grid for spreading spatially the evolving population. Each cell of the mesh contains either one or zero individuals; thus, for instance a 32×32 grid contains up to 1024 individuals, each one placed on a different cell. Each individual is a complete solution to the classification problem, in fact, each individual represents the knowledge that describes the mined dataset. Genetic operators are restricted to immediate neighborhood of the cell in the grid. The size of the neighborhood is r. Given a cell c and $r = 1$, the neighborhood of c is defined by the 8 adjacent cells directly connected to c. Thus, r is the number of hops that defines the neighborhood. Every cell in GALE runs the same algorithm in parallel, without any sequential control, which summarizes as follows:

```
initialize the cell
evaluate classification performance of
        individual in cell
REPEAT
    merge among neighborhood
    split individual in cell
    evaluate classification performance
            of individual in cell
    survival among neighborhood
UNTIL <some end-criterion>
```

The initialization of each cell builds a random individual. Not all the cells contain individuals, thus they can be full (with one individual) or empty. If a cell is not empty, its individual is evaluated using the dataset to be mined. The fitness function used in this paper is very simple, $fit(ind) = \left(\frac{c}{t}\right)^2$ [De Jong and Spears, 1991], where c is the number of correctly classified instances and t the number of instances of the dataset. Next, the evolutionary cycle starts. *Merge* crosses the individual in the cell with one individual randomly chosen among its neighborhood, with a given probability p_m. Merge generates only one individual that replaces the individual in the cell, as later explained. Then, split is applied with a given probability $p_s \cdot fit(ind)$. *Split* copies and mutates the individual in the cell. The new individual is placed in the cell of the neighborhood with higher number of neighbors (occupied cells), or if all cells in the neighborhood are full, in the cell that contains the worst individual. The last step in the evolutionary cycle, *survival*, decides if the individual is kept for the next cycle or not, as explained in the following paragraph. The cell repeats this process until the individual classifies correctly all the instances of the dataset or a certain amount of cycles are run.

The survival step decides if an individual is kept in the cell for the next cycle or not. This process uses the neighborhood information. If a cell has up to one neighbor then the probability of survival of the individual is $p_s^{0,1}(ind_c) = fit(ind)$. Else if a cell has seven or eight neighbors then $p_s^{7,8}(ind_c) = 0$, where the individual is replaced by the best neighbor. In the rest neighborhood configurations, an individual survives if and only if $fit(ind) \geq \overline{\mu}_{nei} + k_{sr} \times \sigma_{nei}$; $\overline{\mu}_{nei}$ is the average fitness value of the occupied neighbor cells, and σ_{nei} their standard deviation. k_{sr} is a parameter that controls the survival pressure over the current cell.

3.2 KNOWLEDGE REPRESENTATION

The evolutionary model of GALE is independent on the knowledge representation evolved. In this paper, GALE evolves one of the three available knowledge representations: if-then rules, instance sets and instance-based decision trees. GALE obtains sets of if-then rules using the codification presented in [De Jong and Spears, 1991], merging individuals with two-point crossover operator and splitting with a mutation operator based on bit inversion. Instance sets are small artificially evolved sets of instances that describe the set of mined instances [Llorà and Garrell, 2001], based on *nearest neighbor* algorithms. *Merge* also uses two-point crossover, and splitting is done using mutation based on generating some new values for genes randomly. The last evolved knowledge representation is based on instance-based decision trees [Llorà and Garrell, 2000b], codified as dynamic trees. The genetic operators used are the basic ones proposed in the Genetic Programming [Koza, 1992] literature.

3.3 SPEEDUP ANALYSIS

Throughout, GALE is a fine-grained parallel model for data mining, independent of the knowledge representation used. As a parallel processing algorithm, the theoretical degree of scalability, or speedup, is a useful measure. In order to compute the theoretical speedup equations, we need to obtain the time complexity equations of both a general-purpose evolutionary algorithm for data mining (GABL) and GALE. In the model equations n is the population size, k the number of iterations of the evolutionary algorithm, l the number of instances in the dataset, r the size of the neighborhood, and p is the number of processors used. The time complexity equations for GABL are:

$$\begin{aligned} t_a &= k \cdot t_{loop} \\ &= k\left(t_{eval} + t_{sel} + t_{cross} + t_{mut}\right) \\ &= k\left(t_{cls}nl + t_{copy}n\log n + t_{xalg}p_x n + t_{malg}p_{mut}n\right) \\ &= kn\left(\alpha_1 l + \alpha_2 \log n + \alpha_3\right) \end{aligned} \tag{1}$$

where t_{cls} is the time of classifying an instance, t_{copy} the time of copying an individual, t_{xalg} the time of the crossover algorithm, and t_{malg} the time of the mutation algorithm. Once we have the time complexity equations of a serial evolutionary algorithm for data mining, $t_a \in O\left(kn\left(l + \log n\right)\right)$, we need the model of GALE. To obtain these equations we assume that each cell of GALE contains one individual, being mapped on a different processor, $n = p$. Each processor is connected to its neighbors in the grid with a latency of $O(1)$. Beneath this assumptions, the equations for

GALE can be calculated as follows,

$$\begin{aligned} t_b^r &= \frac{1}{p} nk \cdot t_{cell} \\ &= \frac{1}{n} nk \cdot t_{cell} \\ &= k\left(t_{eval} + t_{merge} + t_{split} + t_{survival}\right) \\ &= k\left(t_{cls} l + t_{ralg} p_m + t_{salg} p_s + t_{copy}\left(2r+1\right)^2\right) \\ &= k\left(\alpha_1 l + \alpha_2 \left(2r+1\right)^2 + \alpha_3\right) \end{aligned} \tag{2}$$

where t_{cls} is the time of classifying an instance, t_{copy} the time of copying an individual, t_{ralg} the time of the merge algorithm, and t_{salg} the time of splitting. Thus, if $r = 1$ (minimizes communication efforts) then $t_b^1 \in O(kl)$. Finally, the speedup equation becomes:

$$s^1 = \frac{t_a}{t_b^1} = \frac{kn\left(l + \log n\right)}{kl} = n\left(1 + \frac{\log n}{l}\right) \approx n \tag{3}$$

The speedup equation shows that it grows linearly to the number of processors used, avoiding the serial bottleneck selection based on roulette. Equation 2 also shows that the time for mining a data set using GALE does not depend on the population size, it is just a linear function of the number of instances and iterations.

4 TEST SUIT

The test suit designed for evaluating the classification performance of GALE consists of several datasets and machine learning algorithms. This fact lets us study the performance of GALE using statistical tools, like stratified ten-fold cross-validation runs and paired t-tests [Witten and Frank, 2000]. The time performance analysis of GALE as a parallel processing algorithm is beyond the scope of this paper, being part of the further work.

4.1 DATASETS

In order to evaluate the performance of GALE on practice, we performed experiments on twelve datasets. Datasets can be grouped up to three kinds of different datasets: synthetic, public and private. The datasets and their characteristics are listed in table 1.

We used two *synthetic datasets* to tune the learning algorithms, because we knew their solutions in advance: `MX11` is the eleven input multiplexer, and `TAO` is a dataset obtained from sampling the TAO figure using a grid. *Public datasets* are obtained from UCI repository [Merz and Murphy, 1998]. We chose nine datasets from this repository. They are: `breast-w`, `glass`, `ionosphere`, `iris`, `led`, `pima-indians`, `sonar`, and `vehicle`. These datasets contain categorical and numeric attributes, as well as binary an n-ary classification tasks. Finally, we also used two *private datasets* from our own repository. They deal with diagnosis of breast cancer, `biopsies` [Llorà and Garrell, 2000a], and `mammograms` [Llorà and Garrell, 2000b]. `Biopsies` is the result of digitally processing biopsy images, while `mammograms` uses mammographic images.

4.2 CLASSIFIER SCHEMES

As well as GALE algorithm described above, we also run seven additional classifier schemes on the previous datasets. We want to compare the results obtained by GALE with the ones obtained using well-known non-evolutionary classifier schemes. They belong to different learning theories, coded into the *Waikato Environment for Knowledge Analysis* (WEKA) [Witten and Frank, 2000] (available from `http://www.cs.waikato.ac.nz/ml/weka`). We also compare the results with some related evolutionary approaches. Non-evolutionary classifier schemes are:

1. **0-R:** predicts the majority class in the training data. This scheme can be useful for determining a baseline performance as a benchmark for other learning schemes [Witten and Frank, 2000].
2. **IB1 (1-NN):** uses a simple distance measure to find the training instance closest to the given test instance, and predicts the same class as this training instance. If multiple instances have the same (smallest) distance to the test instance, the first one found is used [Aha and Kibler, 1991].
3. **IBk (k-NN):** extends IB1 using a distance measure to find the k training instances closest to the given test instance. It predicts the majority class of the k recovered training instances [Aha and Kibler, 1991].
4. **C4.5 revision 8:** it is based on tree induction. Using the training instances, it induces a decision tree. C4.5r8 is the result of a series of improvements to ID3 [Quinlan, 1986] that include methods for dealing with numeric attributes, missing values and noisy data [Quinlan, 1993].
5. **PART:** induces if-then rules for the given training instances [Frank and Witten, 1986]. PART obtains rules from partially built decision trees.

Table 1: Datasets used for the experiments.

	Dataset	Instances	Missing Values (%)	Noise (%)	Numeric attributes	Nominal attributes	Classes
1	`Biopsies`	1027	0.0	n.a.	24	-	2
2	`Breast-w`	699	0.3	0.0	9	-	2
3	`Glass`	214	0.0	0.0	9	-	6
4	`Ionosphere`	351	0.0	0.0	34	-	2
5	`Iris`	150	0.0	0.0	4	-	3
6	`LED`	2000	0.0	10.0	-	7	10
7	`Mammograms`	216	0.0	n.a.	21	-	2
8	`MX11`	2048	0.0	0.0	-	11	2
9	`Pima-indians`	768	0.0	0.0	8	-	2
10	`Sonar`	208	0.0	0.0	60	-	2
11	`TAO`	1888	0.0	0.0	2	-	2
12	`Vehicle`	846	0.0	0.0	18	-	4

6. **Naive Bayes:** predicts the class for test instances using statistical learning. The core of the system uses the Bayes' rule of conditional probability to achieve the prediction [John and Langley, 1995].
7. **SMO (SVM):** implements the *sequential minimal optimization* algorithm for training a support vector classifier using polynomial kernels [Platt, 1998]. It only performs binary classification tasks.

4.3 RESULTS

GALE was run using the same parameters for all datasets. The grid was 64×64 ($r = 1$) with a 80% of occupied cells after initialization. Other parameters were set to $p_m = 0.4$, $p_s = 0.01$, and $k_{sr} = -0.25$. The maximum number of iterations was 150. The non-evolutionary classifier schemes used the default configuration provided by WEKA. Tables 2 and 3 show the percentage of correct classifications, averaged over stratified ten-fold cross-validation runs. The same folds were used for each scheme. For the LED dataset GALE was run using *hold-out* with a training set of 2000 instances and a test set of 4000 instances. The knowledge representation used in GALE is marked in table 3. A $\star$ marks that GALE evolved if-then rules, † marks instance-based, and ‡ stands for instance-based decision trees.

Table 3 lists the results achieved by GALE. It also marks non-evolutionary schemes with a $\circ$ if they show a significant improvement over the corresponding GALE results, and with a $\bullet$ if they show a significant degradation. Throughout, we speak of results being "significantly different" if the difference is statistically significant at the 1% level according to a paired two-sided t-test, each pair of data points consisting of the estimates obtained in a stratified ten-fold cross-validation run (for the two learning schemes being compared). Table 3 also lists the results of paired two-sided t-tests with a significant level at 10%, just for illustration. Table 4 summarizes the performance of the different methods compared with each other. Numbers indicate how often the method in the row significantly outperforms method in the column. Finally, table 5 presents the performance ranking for the classifier schemes on the test suit.

The results listed in table 4 show that all the classifier schemes clearly outperform the baseline performance obtained by 0-R. GALE obtains significant improvements when compared to IB1, PART, Naive Bayes and SMO. There is no significant improvement when GALE is compared to IBk and C4.5r8, although GALE clearly outperforms them in the overall ranking, table 5, where GALE ends in first place. The first place obtained by GALE shows that it is a robust model for data mining across different domains.

Recently other evolutionary classifier schemes have been applied to data mining. XCS [Wilson, 2000] performs mining on oblique data, been called XCSI. XCSI obtains a mean performance of 95.5±2.89 on the `breast-w`, very similar to 95.7±2.23 obtained by GALE. Inducing oblique decision trees has also been done using evolutionary strategies (OC1-ES) and genetic algorithms (OC1-GA) [Cantú-Paz and Kamath, 2000]. Among other UCI datasets, OC1-ES and OC1-GA were run using `pima-indians` and `iris`. Although the results came from a five-fold cross-validation experiment, OC1-ES obtains 73.7±1.4 and 96.3±1.5 on each dataset, while OC1-GA obtains 73.9±1.3 and 93.6±1.3. GALE outperforms both systems in `pima-indians` dataset (75.65±5.02), being overcame (95.33±3.22) by OC1-ES and OC1-GA in `iris` dataset.

Table 2: Experimental results of the non-evolutionary classifiers for data mining: percentage of correct classifications and standard deviation from stratified ten-fold cross-validation runs.

	Dataset	0-R	IB1	IBk	C4.5r8	PART	NaiveBayes	SMO
1	`Biopsies`	51.61±0.60	83.15±3.15	82.76±4.29	80.04±4.75	79.06±3.27	78.58±5.53	86.36±2.99
2	`Breast-w`	65.51±1.13	95.99±1.45	96.70±1.39	95.42±1.60	95.28±2.15	95.99±2.27	96.71±1.69
3	`Glass`	34.66±2.45	66.35±10.93	66.35±10.93	65.89±10.42	69.15±10.00	47.19±8.95	-
4	`Ionosphere`	64.10±1.09	86.89±4.86	86.32±5.50	89.74±5.02	90.88±3.56	81.76±8.14	87.74±6.22
5	`Iris`	33.33±0.00	95.33±3.22	95.33±3.22	95.33±3.22	95.33±3.22	94.66±2.81	-
6	`LED`	10.45±n.a.	62.40±n.a.	74.95±n.a.	74.90±n.a.	75.12±n.a.	74.85±n.a.	-
7	`Mammograms`	56.01±2.94	62.92±12.42	65.27±6.28	64.81±6.35	62.03±4.16	64.81±7.66	67.12±7.37
8	`MX11`	49.90±0.12	78.61±3.96	99.80±0.25	99.90±0.20	100.00±0.00	61.91±2.65	61.57±2.98
9	`Pima-indians`	65.10±0.96	70.31±3.42	73.83±5.33	73.05±5.22	72.66±5.02	75.39±6.81	76.69±4.60
10	`Sonar`	53.36±3.50	87.50±9.71	82.69±10.37	71.15±8.49	73.08±10.76	67.31±10.82	77.40±8.52
11	`TAO`	49.78±0.16	96.13±1.13	95.97±1.39	95.07±2.02	93.64±2.75	80.77±1.81	83.58±2.25
12	`Vehicle`	25.06±0.53	69.50±5.30	69.73±5.90	73.64±5.26	72.57±4.61	46.21±5.71	-

Table 3: Experimental results and statistical analysis of GALE for data mining: percentage of correct classifications, standard deviation, and results of paired two-sided t-tests across the non-evolutionary classifiers. Each GALE result is marked with the knowledge representation used; a $\star$ for rule sets, a $\dagger$ for instance sets, and a $\ddagger$ for instance-based decision trees.

		0-R	IB1	IBk	C4.5r8	PART	NaiveBayes	SMO	GALE	0-R	IB1	IBk	C4.5r8	PART	NaiveBayes	SMO
	Dataset	two-sided paired t-test (p=0.1)								two-sided paired t-test (p=0.01)						
1	`Biopsies`	●			●	●	●	○	83.64±1.61‡	●				●		
2	`Breast-w`	●							95.70±2.23‡	●						
3	`Glass`	●					●	-	64.95±9.38†	●					●	-
4	`Ionosphere`	●	●	●			●	●	91.46±4.99‡	●						
5	`Iris`	●						-	95.33±3.22†	●						-
6	`LED`	-	-	-	-	-	-	-	74.97±n.a.*	-	-	-	-	-	-	-
7	`Mammograms`	●							66.66±11.55†							
8	`MX11`	●	●	●			●	●	100.00±0.00*	●	●				●	●
9	`Pima-indians`	●	●						75.65±5.02†	●	●					
10	`Sonar`	●			●		●		81.73±9.94†	●					●	
11	`TAO`	●				●	●	●	95.50±1.03†	●					●	●
12	`Vehicle`	●			○	○	●	-	68.79±3.78†	●					●	-

Table 4: Results of paired one-sided t-tests (p=0.01): number indicates how often method in row significantly outperforms method in column.

	0-R	IB1	IBk	C4.5r8	PART	NaiveBayes	SMO	GALE
0-R	-	0	0	0	0	0	0	**0**
IB1	10	-	0	1	2	6	2	**0**
IBk	11	2	-	1	1	5	2	**0**
C4.5r8	11	1	0	-	0	4	2	**0**
PART	11	1	0	0	-	5	2	**0**
NaiveBayes	11	0	0	0	0	-	0	**0**
SMO	8	2	1	1	2	3	-	**1**
GALE	**10**	**2**	**0**	**0**	**1**	**7**	**2**	-

Table 5: Ranking of performance of the different classifier schemes on the test suit.

	`Biopsies`	`Breast-w`	`Glass`	`Ionosphere`	`Iris`	`LED`	`Mammograms`	`MX11`	`Pima-indians`	`Sonar`	`TAO`	`Vehicle`	Avg	rnk
0-R	8	8	7	8	7	7	8	8	8	8	8	7	7.66	8
IB1	3	3	2	5	1	6	6	5	7	1	1	4	3.66	4
IBk	4	2	2	6	1	3	3	4	4	2	2	3	3.00	2
C4.5r8	5	6	4	3	1	4	4	3	5	6	4	1	3.83	6
PART	6	7	1	2	1	1	7	1	6	5	5	2	3.66	4
NaiveBayes	7	4	6	7	6	5	5	6	3	7	7	6	5.75	7
SMO	1	1	-	4	-	-	1	7	1	4	6	-	3.12	3
GALE	2	5	5	1	1	2	2	1	2	3	3	5	2.66	**1**

5 SUMMARY AND CONCLUSIONS

Evolutionary Algorithms have proved to be useful tools for data mining. In this paper, we present a general purpose evolutionary classifier scheme for data mining (GALE). Usually classifier schemes for data mining are tailored for a knowledge representation that fits the dataset to be mined, constraining the scope of the scheme. GALE is not bounded to this constrain because it is knowledge independent, thus it can evolve if-then rules, instance sets, or instance-based decision trees. Another important point in data mining is the real time spent in the mining process. Evolutionary algorithms tend to be time-consuming. GALE can reduce the amount of real time required exploiting fine-grained parallel processing. The classifier scheme is based on spatial neighborhood relations on a 2D grid, where local genetic operators and massive parallel fitness computation are defined. The time complexity of GALE is independent of the population size, being only bounded by the number of iterations and the size of the dataset to be mined, $O(kl)$.

GALE and other seven well-known classifier schemes were run using twelve different datasets. The obtained results show that GALE, although its simplicity, is a competitive classifier scheme for data mining. Results also show its robustness when applied to very different datasets compared to other classifier schemes, including evolutionary ones. Nevertheless, these results are only a first step into the data mining tasks. Further work includes a deeper analysis of the behavior of GALE on other data mining tasks, as well as a deeper study of the time performance of the parallel processing done by GALE.

Acknowledgments

The authors would like to thank the support provided by *CIRIT* and *Fondo de Investigación Sanitaria - Instituto Carlos III-* (under grant numbers 1999FI-00719 and FIS-00/0033-2), as well as the support of *Enginyeria i Arquitectura La Salle* to our research group. We are also grateful to Ian H. Witten and Eibe Frank for providing their codes on-line, and to all the people who donated the datasets. Finally, we want to thank Erick Cantú-Paz for his useful comments during the preparation of this paper.

References

[Aha and Kibler, 1991] Aha, D. and Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning, Vol. 6*, pages 37–66.

[Araujo et al., 1999] Araujo, D. L., Lopes, H. S., and Freitas, A. A. (1999). A Parallel Genetic Algorithm for Rule Discovery in Large Databases. In *IEEE Systems, Man and Cybernetics Conference*, volume III, pages 940–945.

[Araujo et al., 2000] Araujo, D. L., Lopes, H. S., and Freitas, A. A. (2000). Rule Discovery with a Parallel Genetic Algorithms. *Workshop on Data Mining with Evolutionary Computation held in GECCO2000*, pages 89–92.

[Cantú-Paz, 1999] Cantú-Paz, E. (1999). Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms. *Genetic and Evolutionary Computation Conference (GECCO99)*, pages 91–98.

[Cantú-Paz and Kamath, 2000] Cantú-Paz, E. and Kamath, C. (2000). Using Evolutionary Algorithms to Induce Oblique Decision Trees. *Genetic and Evolutionary Computation Conference (GECCO2000)*, pages 1053–1060.

[De Jong and Spears, 1991] De Jong, K. A. and Spears, W. M. (1991). Learning Concept Classification Rules Using Genetic Algorithms. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 651–656. Morgan Kaufmann Publishers.

[Flockhart, 1995] Flockhart, I. W. (1995). GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithms (Final Report). Technical Report EPCC-AIKMS-GA-MINER-REPORT 1.0, University of Edinburgh.

[Frank and Witten, 1986] Frank, E. and Witten, I. H. (1986). Generating Accurate Rule Sets Without Global Optimization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 152–160. Kluber.

[Freitas and Lavinston, 1998] Freitas, A. A. and Lavinston, S. H. (1998). *Mining Very Large Databases with Parallel Processing.* Kluwer Academic Publishers.

[Gordon and Whitley, 1993] Gordon, V. S. and Whitley, D. (1993). Serial and Parallel Genetic Algorithms as Function Optimizers. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 177–183. Morgan Kaufmann Publishers.

[Hwang, 1993] Hwang, K. (1993). *Advanced Computer Architectures: parallelism, scalability, programability.* MacGraw-Hill.

[John and Langley, 1995] John, G. H. and Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufman Publishers.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programing: On the Programing of Computers by Means of Natural Selection (Complex Adaptive Systems).* MIT Press.

[Llorà and Garrell, 2000a] Llorà, X. and Garrell, J. M. (2000a). Automatic Classification and Artificial Life Models. In *Proceedings of Learning00 Workshop.*

[Llorà and Garrell, 2000b] Llorà, X. and Garrell, J. M. (2000b). Evolving Hierarchical Agents using Cellular Genetic Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO2000)*, page 868. Morgan Kaufmann Publishers.

[Llorà and Garrell, 2001] Llorà, X. and Garrell, J. M. (2001). Inducing partially-defined instances with Evolutionary Algorithms. In *Proceedings of the 18th International Conference on Machine Learning (*to appear*).*

[Manderick and Spiessens, 1989]
Manderick, B. and Spiessens, P. (1989). Fine-Grained Parallel Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 428–433. Morgan Kaufmann Publishers.

[Merz and Murphy, 1998] Merz, C. J. and Murphy, P. M. (1998). UCI Repository for Machine Learning DataBases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. *Irvine, CA: University of California, Department of Information and Computer Science.*

[Mühlenbein, 1989] Mühlenbein, H. (1989). Parallel Genetic Algorithms and Combinatorial Optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann Publishers.

[Pettey et al., 1987] Pettey, C. B., Leuze, M. R., and Grefenstette, J. J. (1987). A Parallel Genetic Algorithm. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 155–161. Lawerence Erlbaum Associates Publishers.

[Platt, 1998] Platt, J. C. (1998). Fast Training of Support Vector Machines using Sequential Minimal Optimization. *Advances in Kernel Methods - Support Vector Learning.*

[Povinelli, 2000] Povinelli, R. J. (2000). Using Genetic Algorithms to find Temporal Patterns Indicative of Time Series Events. *Workshop on Data Mining with Evolutionary Computation held in GECCO2000*, pages 80–84.

[Quinlan, 1986] Quinlan, R. (1986). Induction of decission trees. *Machine Learning, Vol. 1, No. 1*, pages 81–106.

[Quinlan, 1993] Quinlan, R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers.

[Robertson, 1987] Robertson, G. G. (1987). Parallel implementation of Genetic Algorithms in a Classifier System. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 155–161. Lawerence Erlbaum Associates Publishers.

[Spiessens and Manderick, 1991] Spiessens, P. and Manderick, B. (1991). A Massively Parallel Genetic Algorithms: implementation and first analysis. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 279–285. Morgan Kaufmann Publishers.

[Whitley, 1993] Whitley, D. (1993). Cellular Genetic Algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, page 658. Morgan Kaufmann Publishers.

[Wilson, 1995] Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175.

[Wilson, 2000] Wilson, S. W. (July, 2000). Mining Oblique Data with XCS. *IlliGAL Report No. 2000028.*

[Witten and Frank, 2000] Witten, I. H. and Frank, E. (2000). *DataMining: practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann Publishers.