GENETIC ALGORITHMS

Keith Mathias, chair

Partnering Strategies for Fitness Evaluation in a Pyramidal Evolutionary Algorithm

Uwe Aickelin
Intelligent Computer Systems Centre
University of the West of England
Bristol BS16 1QY, UK

Larry Bull
Intelligent Computer Systems Centre
University of the West of England
Bristol BS16 1QY, UK

Abstract

This paper combines the idea of a hierarchical distributed genetic algorithm with different inter-agent partnering strategies. Cascading clusters of sub-populations are built from bottom up, with higher-level sub-populations optimising larger parts of the problem. Hence higher-level sub-populations search a larger search space with a lower resolution whilst lower-level sub-populations search a smaller search space with a higher resolution. The effects of different partner selection schemes for (sub-)fitness evaluation purposes are examined for two multiple-choice optimisation problems. It is shown that random partnering strategies perform best by providing better sampling and more diversity.

1 INTRODUCTION

When hierarchically distributed evolutionary algorithms are combined with multi-agent structures a number of new questions become apparent. One of these questions is addressed in this paper: the issue of assigning a meaningful (sub-) fitness to an agent. This paper will look at seven different partnering strategies for fitness evaluation when combined with a genetic algorithm that uses a co-operative sub-population structure. We will evaluate the different strategies according to their optimisation performance of two scheduling problems.

Genetic algorithms are generally attributed to Holland [1976] and his students in the 1970s, although evolutionary computation dates back further (refer to Fogel [1998] for an extensive review of early approaches). Genetic algorithms are stochastic meta-heuristics that mimic some features of natural evolution. Canonical genetic algorithms were not intended for function optimisation, as discussed by De Jong [1993]. However, slightly modified versions proved very successful. For an introduction to genetic algorithms for function optimisation, see Deb [1996].

The twist when applying our type of distributed genetic algorithm lies in its special hierarchical structure. All sub-populations follow different (sub-) fitness functions, so in effect only searching specific parts of the solution space. Following special crossover-operators these parts are then gradually merged to full solutions. The advantage of such a divide and conquer approach is reduced epistasis within the lower-level sub-populations which makes the optimisation task easier for the genetic algorithm.

The paper is arranged as follows: the following section describes the nurse scheduling and tenant selection problems. Pyramidal genetic algorithms and their application to these two problems are detailed in section 3. Section 4 explains the seven partnering strategies examined in the paper and section 5 describes their use and computational results. The final section discusses all findings and draws conclusions.

2 THE NURSE SCHEDULING PROBLEM

Two optimisation problems are considered in this paper, the nurse scheduling problem and the tenant selection problem. Both have a number of characteristics that make them an ideal testbed for the enhanced genetic algorithm using partnering strategies. Firstly, they are both in the class of NP-complete problems [Johnson 1998, Martello & Toth 1990]; hence, they are challenging problems. Secondly, they have proved resistant to optimisation by a standard genetic algorithm, with good solutions only found by using a novel strategy of indirectly optimising the problem with a decoder based genetic algorithm [Aickelin & Dowsland 2001]. Finally, both problems are similar multiple-choice allocation problems. For the nurse scheduling, the choice is to allocate a shift-pattern to each

nurse, whilst for the tenant selection it is to allocate an area of the mall to a shop. However, as the following more detailed explanation of the two will show, the two problems also have some very distinct characteristics making them different yet similar enough for an interesting comparison of results.

The nurse-scheduling problem is that of creating weekly schedules for wards of up to 30 nurses at a major UK hospital. These schedules have to satisfy working contracts and meet the demand for given numbers of nurses of different grades on each shift, whilst at the same time being seen to be fair by the staff concerned. The latter objective is achieved by meeting as many of the nurses' requests as possible and by considering historical information to ensure that unsatisfied requests and unpopular shifts are evenly distributed. Due to various hospital policies, a nurse can normally only work a sub-set of the 411 theoretically possible shift-patterns. For instance, a nurse should work either days or nights in a given week, but not both. The interested reader is directed to Aickelin & Dowsland [2000] and Dowsland [1998] for further details of this problem.

For our purposes, the problem can be modelled as follows. Nurses are scheduled weekly on a ward basis such that they work a feasible pattern with regards to their contract and that the demand for all days and nights and for all qualification levels is covered. In total three qualification levels with corresponding demand exist. It is hospital policy that more qualified nurses are allowed to cover for less qualified one. Infeasible solutions with respect to cover are not acceptable. A solution to the problem would be a string, with the number of elements equal to the number of nurses. Each element would then indicate the shift-pattern worked by a particular nurse. Depending on the nurses' preferences, the recent history of patterns worked and the overall attractiveness of the pattern, a penalty cost is then allocated to each nurse-shift-pattern pair. These values were set in close consultation with the hospital and range from 0 (perfect) to 100 (unacceptable), with a bias to lower values. The sum of these values gives the quality of the schedule. 52 data sets are available, with an average problem size of 30 nurses per ward and up to 411 possible shift-patterns per nurse.

The problem can be formulated as an integer linear program as follows.

Indices:

i = 1...n nurse index.

j = 1...m shift pattern index.

k = 1...7 are days and 8...14 are nights.

s = 1...p grade index.

Decision variables:

$$x_{ij} = \begin{cases} 1 & \text{nurse } i \text{ works shift pattern } j \\ 0 & \text{else} \end{cases}$$

Parameters:

n = Number of nurses.

m = Number of shift patterns.

p = Number of grades.

$$a_{jk} = \begin{cases} 1 & \text{shift pattern } j \text{ covers day / night } k \\ 0 & \text{else} \end{cases}$$

$$q_{is} = \begin{cases} 1 & \text{nurse } i \text{ is of grade } s \text{ or higher} \\ 0 & \text{else} \end{cases}$$

p_{ij} = Preference cost of nurse i working shift pattern j.

N_i = Shifts per week of nurse i if night shifts are worked.

D_i = Shifts per week of nurse i if day shifts are worked.

B_i = Shifts per week of nurse i if both are worked.

R_{ks} = Demand of nurses with grade s on day or night k.

$F(i)$ = Set of feasible shift patterns for nurse i, defined as

$$F(i) = \left\{ \begin{array}{ll} \sum_{k=1}^{7} a_{jk} = D_i & \forall j \in \textit{day shifts} \\ or & \\ \sum_{k=8}^{14} a_{jk} = N_i & \forall j \in \textit{night shifts} \\ or & \\ \sum_{k=1}^{14} a_{jk} = B_i & \forall j \in \textit{combined shifts} \end{array} \right\} \forall i$$

Target function:

$$\sum_{i=1}^{n} \sum_{j \in F(i)}^{m} p_{ij} x_{ij} \quad \rightarrow \quad \text{min!}$$

Subject to:

$$\sum_{j \in F(i)} x_{ij} = 1 \qquad \forall i \qquad (1)$$

$$\sum_{j \in F(i)} \sum_{i=1}^{n} q_{is} a_{jk} x_{ij} \geq R_{ks} \qquad \forall k, s \qquad (2)$$

Constraint set (1) ensures that every nurse works exactly one shift pattern from his/her feasible set, and constraint set (2) ensures that the demand for nurses is covered for every grade on every day and night. Note that the definition of q_{is} is such that higher graded nurses can substituted those at lower grades if necessary. Typical problem dimensions are 30 nurses of three grades and 411 shift patterns. Thus, the Integer Programming formulation has about 12000 binary variables and 100 constraints.

Finally for all decoders, the fitness of completed solutions has to be calculated. Unfortunately, feasibility cannot be guaranteed, as otherwise an unlimited supply of nurses, respectively overtime, would be necessary. This is a problem-specific issue and cannot be changed. Therefore, we still need a penalty function approach. Since the chosen encoding automatically satisfies constraint set (1) of the integer programming formulation, we can use the following formula, where w_{demand} is the penalty weight, to calculate the fitness of solutions. Hence the penalty is proportional to the number of uncovered shifts and the fitness of a solution is calculated as follows.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}x_{ij} + w_{demand}\sum_{k=1}^{14}\sum_{s=1}^{p}\max\left[R_{ks} - \sum_{i=1}^{n}\sum_{j=1}^{m} q_{is}a_{jk}x_{ij}; 0\right] \rightarrow \min$$

Here we use an encoding that follows directly from the Integer Programming formulation. Each individual represents a full one-week schedule, i.e. it is a string of n elements with n being the number of nurses. The *ith* element of the string is the index of the shift pattern worked by nurse i. For example, if we have 5 nurses, the string (1,17,56,67,3) represents the schedule in which nurse 1 works pattern 1, nurse 2 pattern 17 etc.

For comparison, all data sets were attempted using a standard Integer Programming package [Fuller 1998]. However, some remained unsolved after each being allowed 15 hours run-time on a Pentium II 200. Experiments with a number of descent methods using different neighbourhoods, and a standard simulated annealing implementation, were even less successful and frequently failed to find feasible solutions. A straightforward genetic algorithm approach failed to solve the problem [Aickelin & Dowsland 2000]. The best evolutionary results to date have been achieved with an indirect genetic approach employing a decoder function [Aickelin & Dowsland 2001]. However, we believe that there is further leverage in direct evolutionary approaches to this problem. Hence, we propose to use an enhanced pyramidal genetic algorithm in this paper.

3 TENANT SELECTION PROBLEM

The second problem is a mall layout and tenant selection problem; termed the mall problem here. The mall problem arises both in the planning phase of a new shopping centre and on completion when the type and number of shops occupying the mall has to be decided. To maximise revenue a good mixture of shops that is both heterogeneous and homogeneous has to be achieved. Due to the difficulty of obtaining real-life data because of confidentiality, the problem and data used in this research are constructed artificially, but closely modelled after the actual real-life problem as described for instance in Bean et al. [1988]. In the following, we will briefly outline our model.

The objective of the mall problem is to maximise the rent revenue of the mall. Although there is a small fixed rent per shop, a large part of a shop's rent depends on the sales revenue generated by it. Therefore, it is important to select the right number, size and type of tenants and to place them into the right locations to maximise revenue. As outlined in Bean et al. [1988], the rent of a shop depends on the following factors:

- The attractiveness of the area in which the shop is located.
- The total number of shops of the same type in the mall.
- The size of the shop.
- Possible synergy effects with neighbouring similar shops, i.e. shops in the same group (not used by Bean et al.).
- A fixed amount of rent based on the type of the shop and the area in which it is located.

This problem can be modelled as follows: Before placing shops, the mall is divided into a discrete number of locations, each big enough to hold the smallest shop size. Larger sizes can be created by placing a shop of the same type in adjacent locations. Hence, the problem is that of placing i shop-types (e.g. menswear) into j locations, where each shop-type can belong to one or more of l groups (e.g. clothes shops) and each location is situated in one of k areas. For each type of shop there will be a minimum, ideal and maximum number allowed in the mall, as consumers are drawn to a mall by a balance of variety and homogeneity of shops.

The size of shops is determined by how many locations they occupy within the same area. For the purpose of this study, shops are grouped into three size classes, namely small, medium and large, occupying one, two and three locations in one area of the mall respectively. For instance, if there are two locations to be filled with the

same shop-type within one area, then this will be a shop of medium size. If there are five locations with the same shop-type assigned in the same area, then they will form one large and one medium shop etc. Usually, there will also be a maximum total number of small, medium and large shops allowed in the mall.

To test the robustness and performance of our algorithms thoroughly on this problem, 50 problem instances were created. All problem instances have 100 locations grouped into five areas. However, the sets differ in the number of shop-types available (between 50 and 20) and in the tightness of the constraints regarding the minimum and maximum number of shops of a certain type or size. Full details of the model and how the data was created, its dimensions and the differences between the sets can be found in [Aickelin 1999].

4 PYRAMIDAL GENETIC ALGORITHMS

Both problems failed to be optimised with a standard genetic algorithm [Aickelin & Dowsland 2000, 2001]. Our previous research showed that the difficulties were attributable to epistasis created by the constrained nature of the optimisation. Briefly, epistasis refers to the 'non-linearity' of the solution string [Davidor 1991], i.e. individual variable values which were good in their own right, e.g. a particular shift / location for a particular nurse / shop formed low quality solutions once combined. This effect was created by those constraints that could only be incorporated into the genetic algorithm via a penalty function approach. For instance, most nurses preferred working days; thus, partial solutions with many 'day' shift-patterns have a higher fitness. However, combining these shift-patterns leads to shortages at night and therefore infeasible solutions. The situation for the mall problem is similar yet more complex, as two types of constraints have to be dealt with: size constraints and number constraints.

In [Aickelin & Dowsland 2000] we presented a simple, and on its own unsuccessful, pyramidal genetic algorithm for the nurse-scheduling problem. A pyramidal approach can best be described as a hierarchical coevolutionary genetic algorithm where cascading clusters of sub-populations are built from bottom up. Higher-level sub-populations have individuals with loner strings and optimise larger parts of the problem. Thus, the hierarchy is not within one string but rather between sub-populations which optimise different problem portions. Hence, higher-level sub-populations search a larger search space with a lower resolution whilst lower-level sub-populations search a smaller search space with a higher resolution. A related hierarchical framework was presented using Genetic Programming [Koza 1991] whereby main program trees coevolve with successively lower level functions [e.g. Ahluwalia & Bull 1998]. The pyramidal GA can be applied to the nurse-scheduling problem in the following way:

- Solutions in sub-populations 1, 2 and 3 have their fitness based on cover and requests only for grade 1, 2 and 3 respectively.
- Solutions in sub-populations 4, 5 and 6 have their fitness based on cover and requests for grades (1+2), (2+3) and (3+1).
- Solutions in sub-population 7 optimise cover and requests for (1+2+3).
- Solutions in sub-population 8 solve the original (all) problem, i.e. cover for 1, for (1+2) and for (1+2+3).

The full structure is illustrated in figure 1. Sub-solution strings from lower populations are cascaded upwards using suitable crossover and selection mechanisms. For instance, fixed crossover points are used such that a solution from sub-population (1) combined with one from (1+2) forms a new solution in sub-population (1+2). Each sub-population performs 50% of crossovers uniform with two parents from itself. The other 50% are done by taking one parent from itself and the other from a suitable lower level population and then performing a fixed-point crossover. Bottom level sub-populations use only uniform crossover. The top level (all) population randomly chooses the second parent from all other populations. Although the full problem is as epistatic as before, the sub-problems are less so as the interaction between nurse grades is (partially) ignored. Compatibility problems of combining the parts are reduced by the pyramidal structure with its hierarchical and gradual combining. This can be seen as similar to the "Island Injection" parallel GA system [Eby et al. 1999].

Using this approach improved solution quality in comparison to a standard genetic algorithm was recorded. Initially roulette wheel selection based on fitness rank had been used to choose parents. The fitness of each sub-string is calculated using a substitute fitness measure based on the requests and cover as detailed above, i.e. the possibility of more qualified nurses covering for less-qualified ones is partially ignored. Unsatisfied constraints are still included via a penalty function. This paper will investigate various partnering strategies between the agents of the sub-populations to improve upon these results.

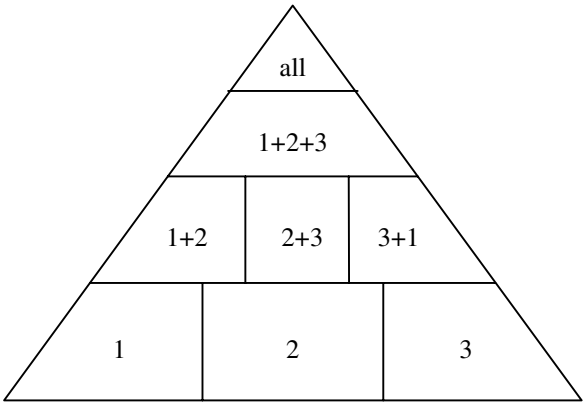


Figure 1: Nurse Problem Pyramidal Structure.

Similar to the nurse problem, a solution to the mall problem can be represented by a string with as many elements as locations in the mall. Each element then indicates what shop-type is to be located there. The mall is geographically split into different regions, for instance north, east, south, west and central. Some of the objectives are regional; e.g. the size of a shop, the synergy effects, the attractiveness of an area to a shop-type, whereas others are global, e.g. the total number of shops of a certain type or size.

The application of the pyramidal structure to the mall problem follows along similar lines to that of the nurse problem. In line with decomposing partitions into those with nurses of the same grade, the problem is now split into the areas of the mall. Thus, we will have sub-strings with all the shops in one area in them. These can then be combined to create larger 'parts' of the mall and finally full solutions.

However, the question arises how to calculate the substitute fitness measure of the partial strings. The solution chosen here will be a pseudo measure based on area dependant components only, i.e. global aspects are not taken into account when a substitute fitness for a partial string is calculated. Thus, sub-fitness will be a measure of the rent revenue created by parts of the mall, taking into account those constraints that are area based. All other constraints are ignored. A penalty function is used to account for unsatisfied constraints.

Due to the complexity of the fitness calculations and the limited overall population size, we refrained from using several levels in the hierarchical design as we did with the nurse scheduling. Instead a simpler two-level hierarchy is used as shown in figure 2: Five sub-populations optimising the five areas separately (1,2,3,4,5) and one main population optimising the original problem (all). Within the sub-populations 1-5 uniform crossover is used. The top-level population uses uniform crossover between two members of the population half the time and for the remainder a special crossover that selects one solution from a random sub-population that then performs a fixed-point crossover with a member of the top population.

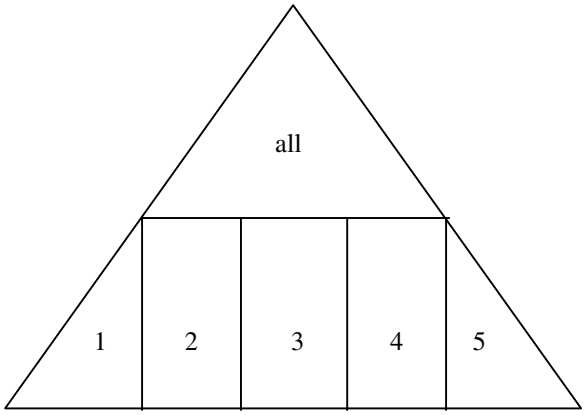


Figure 2: Mall Problem Pyramidal Structure.

The remainder of this paper will investigate ways to try to improve on previously found poor results by suggesting ways of combining partial strings more intelligently. An alternative, particularly for the mall problem, would be a more gradual build-up of sub-populations. Without increasing the overall population size, this would lead to more and hence smaller sub-populations. However, this more gradual approach might have enabled the algorithm to find good feasible solutions by more slowly joining together promising building blocks. This is in contrast to the relatively harsh two-level and three-level design where building blocks had to 'succeed' immediately. Exploring the exact benefits of a gradual build-up of sub-solutions would make for another challenging area of possible future research.

5 PARTNERING STRATEGIES

The problem of how to pick partners has been noted in both competitive and co-operative coevolutionary algorithms. Many strategies have been presented in the literature as summarised for instance in [Bull 1997]. In this paper, the following strategies are compared for their effectiveness in fighting epistasis by giving meaningful (sub-) fitness values in the pyramidal genetic algorithm optimising the nurse scheduling and the mall problems.

- Rank-Selection (S): This is the method used so far in our algorithms. Solutions are assigned a sub-fitness

score based as closely as possible on the contribution of their partial string to full solutions. All solutions are then ranked within each sub-population and selection follows a roulette wheel scheme based on the ranks [e.g. Aickelin & Dowsland 2000].

- Random (R): Solutions choose their mating partners randomly from amongst all those in the sub-population their sub-population is paired with [e.g. Bull & Fogarty 1993].
- Best (B): In this strategy, each agent is paired with the current best solution of the other sub-population(s). In case of a tie, the solution with the lower population index is chosen [e.g. Potter & De Jong 1994].
- Distributed (D): The idea behind this approach is to match solutions with similar ones to those paired with previously [e.g. Ackley & Littman 1994]. To achieve this each sub-population is spaced out evenly across a single toroidal grid. Subsequently, solutions are paired with others on the same grid location in the appropriate other sub-populations. Children created by this are inserted in an adjacent grid location. This is said to be beneficial to the search process because a consistent coevolutionary pressure emerges since all offspring appear in their parents' neighbourhoods [Husbands 1994]. In our algorithms, we use local mating with the neighbourhood set to the eight agents surrounding the chosen location.
- Best / Random (BR): A solution is paired twice: with the best of the other sub-population(s) and with a random partner(s). The better of the two fitness values is recorded.
- Rank-based / Random (SR): A solution is paired twice: with roulette wheel selected solution(s) and with (a) random partner(s). The better of the two fitness values is recorded.
- Random / Random (RR): A solution is paired twice with random partner(s). The better of the two fitness values is recorded.

6 EXPERIMENTAL RESULTS

6.1 THE MODEL

To allow for fair comparison, the parameters and strategies used for both problems are kept as similar as possible. Both have a total population of 1000 agents. These are split into sub-populations of size 100 for the lower-levels and a main population of size 300 for the nurse scheduling and respectively of size 500 for the mall problem. In principle, two types of crossover take place: within sub-populations a two-parent-two-children parameterised uniform crossover with p=0.66 for genes coming from one parent takes place.

Each new solution created undergoes mutation with a 1% bit mutation probability, where a mutation would re-initialise the bit in the feasible range. The algorithm is run in generational mode to accommodate the sub-population structure better. In every generation the worst 90% of parents of all sub-populations are replaced. For all fitness and sub-fitness function calculations a fitness score as described before is used. Constraint violations are penalised with a dynamic penalty parameter, which adjusts itself depending on the (sub)-fitness difference between the best and the best feasible agent in each (sub-) population. Full details on this type of weight and how it was calculated can be found in Smith & Tate [1993] and Aickelin & Dowsland [2000]. The stopping criterion is the top sub-population showing no improvement for 50 generations.

To obtain statistically sound results all experiments were conducted as 20 runs over all problem instances. All experiments were started with the same set of random seeds, i.e. with the same initial populations. The results are presented in feasibility and cost respectively rent format. Feasibility denotes the probability of finding a feasible solution averaged over all problem instances. Cost / Rent refer to the objective function value of the best feasible solution for each problem instance averaged over the number of instances for which at least one feasible solution was found.

Should the algorithm fail to find a single feasible solution for all 20 runs on one problem instance, a censored observation of one hundred in the nurse case and zero for the mall problem is made instead. As we are minimising the cost for the nurses and maximising the rent of the mall, this is equivalent to a very poor solution. For the nurse-scheduling problem, the cost represents the sum of unfulfilled nurses' requests and unfavourable shift-patterns worked. For the mall, the values for the rent are in thousands of pounds per year.

6.2 RESULTS

Table 1 shows the results for a variety of fitness evaluation strategies used and compares these to the theoretic bounds (Bound) and the standard genetic algorithm approach (SGA). For the Nurse Scheduling Problem all strategies used give better results than those found by the SGA. However, as explained above, most credit for this is attributed to the pyramidal structure reducing epistasis.

On closer examination, rank-based (S), random (R) and distributed (D) perform almost equally well, with the rank-based method being slightly better than the other two. All three methods have in common that they contain

a stochastic element in the choice of partner. The benefit of this is apparent when compared to the best (B) method. Here the results are far worse which we attributed to the inherently restricted sampling. Interestingly, using the double schemes (SR, BR and RR) improves results across the board, which again strengthens our hypothesis how important good sampling is. The overall best results are found by the double random (RR) method. These results correspond to those reported in [Bull 1997].

The results for the Mall problem are similar to those found for the nurse problem: Double strategies work better than single ones and the Best strategy does particularly poorly. However, unlike for the nurse scheduling none of the single strategies significantly improves results over the SGA approach. Reasons for this have already been outlined in the previous sections, i.e. mainly the nature of splitting the problem into sub-problems being contrary to many of the problem's constraints. On the other hand, even for the simple strategies results are far improved over those found by using the partnering strategies for mating, whilst those found by the double strategies even outperform the SGA. We believe that this can be explained as follows: The main downfall of the partnering for mating strategies for the mall problem was outside control of these strategies. It lies in the fact that the sub-fitness scores are not a good predictor for the success of sub-solutions. However, as these results show, if the original (sub-)fitness measures are substituted by full fitness scores based on good partnering methods the pyramidal structure does work. This confirms our suspicion that the previous 'failure' of the pyramidal idea for the mall problem was rooted within our choice of sub-fitness measures rather than in the hierarchical sub-population idea itself.

Method	N Cost	N Feasibility	M Rent	M Feasibility
Bound	8.8	100%	2640	100%
SGA	54.2	33%	1850	94%
S	13.3	79%	1860	90%
R	14.5	77%	1915	94%
B	35.9	44%	1550	72%
D	14.6	77%	1820	88%
SR	12.7	84%	1950	99%
BR	14.2	81%	1897	86%
RR	12.1	83%	1955	99%

Table 1: Partnering Strategies for Fitness Evaluation Results (N = Nurse, M = Mall).

6.3 NURSE SCHEDULING WITH A HILLCLIMBER

The results presented so far show that even with the best algorithm for the nurse scheduling problem some data instances were unsolvable. In order to overcome this, a special hillclimber has been developed which is fully described in [Aickelin & Dowsland 2001]. The use of local search to refine solutions produced via the GA for complex problem domains is well established – often termed memetic algorithms [e.g. Moscato 1999]. Briefly, the hill-climber is local search based algorithm that iteratively tries to improve solutions by (chain-) swapping shift patterns between nurses or alternatively assigns a strictly solution improving pattern to a nurse. As the hill climber is computationally expensive, it is only used on those solutions showing favourable characteristics for it to exploit. Those solutions are referred to as 'balanced' and one example is a nurse surplus on one day shift and a shortage on another day shift.

The last set of experiments presented in table 2 shows what impact the best partnering schemes for evaluation (RR) has once the previously excluded hillclimber is attached to the genetic algorithm. The results reveal that the SGA is outperformed by the double random fitness evaluation approach coupled with the hill climber. One possible explanation for this effect can be found by having a closer look at the RR operator. Gains are most likely made due to better sampling. However, as mentioned before there is a large stochastic element involved in this case. Judging from these results it seems that this is beneficial as it leads to a bigger variety of solutions in turn leaving more for the hill climber to exploit.

Algorithm	Short	N Cost	N Feasibility
SGA & Hillclimber	SGA&H	10.8	91%
RR & Hillclimber	RR&H	9.9	95%

Table 2: Results for Algorithms combined with a Hillclimber for the Nurse Scheduling Problem.

7 CONCLUSIONS

Using the partnering strategies for evaluation purposes yields results in accordance with those reported in [Bull 1997]. For both problems the simple strategies worked equally well apart from the restricting 'best' choice. Combining two partnering schemes improved results

further with the overall best solutions found by the double random strategy. Interestingly, the improvements of results seemed to be based on better sampling and more diversity. Thus for this approach an additional hillclimber is able to improve solutions beyond the previously best ones.

REFERENCES

Ackley D H & Littman M L (1994) “Altruism in the Evolution of Communication”, in R Brooks & P Maes (Eds.) Artificial Life IV, MIT Press, Mass., pp 40-48.

Ahluwalia M. & Bull L. (1998) “Coevolving Functions in Genetic Programming: Dynamic ADF Creation using GliB”. In V.W. Porto, N. Saravanan, D. Wagen & A.E. Eiben (eds.) Proceedings of the Seventh Annual Conference on Evolutionary Programming. Springer Verlag, pp 809-818.

Aickelin U (1999). “Genetic Algorithms for Multiple-Choice Optimisation Problems.” PhD Dissertation, University of Wales, Swansea, United Kingdom.

Aickelin U and Dowsland K (2000). “Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem.” Journal of Scheduling 3, pp 139-153.

Aickelin U and Dowsland K (2001). “An indirect genetic algorithm approach to a nurse scheduling problem.” Under review by the Journal of Computing and Operational Research.

Bean J, Noon C, Ryan S, Salton G (1988), “Selecting Tenants in a Shopping Mall,” Interfaces 18, pp 1-9.

Bull, L. (1997) Evolutionary Computing in Multi-Agent Environments: Partners. In T. Baeck (ed.) Proceedings of the Seventh International Conference on Genetic Algorithms. Morgan Davis, L. (1991) Handbook of Genetic Algorithms. Van Nostrand Reinhold. Kaufmann, pp 370-377.

Bull L & Fogarty T C (1993), “Coevolving Communicating Classifier Systems for Tracking”, in R F Albrecht, C R Reeves & N C Steele (eds.) Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, New York, pp 522-527.

Deb K (1996), Genetic Algorithms for Function Optimisation, in F. Herrera, J.L. Verdegay (Editors), Studies in Fuzziness and Soft Computing Volume 8, Physica Verlag, pp 4-31.

Davidor Y. (1991), Epistasis Variance: A Viewpoint on GA-Hardness, Foundations of Genetic Algorithms 1, 23-35, G Rawlins (Ed), Morgan Kaufmann, 1991.

Eby D, Averill R, Goodman E & Punch W (1999) "Optimal Design of Flywheels Using an Injection Island Genetic Algorithm” Artificial Intelligence Engineering Design, Analysis and Manufacturing, 13, pp. 389-402.

Fuller E. (1998), Tackling Scheduling Problems Using Integer Programming. Master Thesis, University of Wales Swansea, United Kingdom, 1998.

Holland J. (1976), Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1976.

Husbands P (1994), “Distributed Coevolutionary Genetic Algorithms for Multi-Criteria and Multi-Constraint Optimisation”, in T C Fogarty (ed.) Evolutionary Computing, Springer-Verlag, pp 150-165.

Iba H (1996), “Emergent Co-operation for Multiple Agents Using Genetic Programming”, in H-M Voigt, W Ebeling, I Rechenberg & H-P Schwefel (eds.) Parallel Problem Solving from Nature - PPSN IV, Springer, Berlin, pp 32-41.

Johnson D.S. (1998), private communication, 1998.

Koza, J.R. (1991) Genetic Programming. MIT Press.

Martello S. and Toth P. (1990), Knapsack Problems, Wiley, Chichester, 1990.

Moscato P (1999), Memetic Algorithms: A Short Introduction, in New Ideas in Optimization, Corne D, Dorigo M & Glover F (eds), Mc Graw Hill, pp 219-234, 1999.

Potter M. & De Jong K. (1994) A Co-operative Coevolutionary Approach to Function Optimisation. In Y Davidor, H-P Schwefel & R Manner (eds.) Parallel Problem Solving From Nature - PPSN III, Springer-Verlag, Berlin, pp 249-259.

Ronald E (1995), When Selection Meets Seduction, pp 167-173 in Eshelman L, Proceedings of the International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Francisco, 1995.

Smith A. and Tate D. (1993), Genetic Optimisation Using a Penalty Function, Proceedings ICGA 5, pp 499-505, Forrest S (ed.), Morgan Kaufmann, 1993.

Stanley A E, Ashlock D, Testatsion L (1994), “Iterated Prisoner’s Dilemma with Choice and Refusal of Partners”, in C G Langton (ed.) Artificial Life III, Addison-Wesley, Redwood City, pp 131-146.

Wolpert D & Macready W (1995), No Free Lunch Theorem for Search, SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, pp 1-38, 1995.

Efficient Discretization Scheduling in Multiple Dimensions

Laura A. Albert
Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign
1206 W. Green St.
Urbana, IL 61801
(217) 244-0292
lalbert@uiuc.edu

David E. Goldberg
Department of General Engineering
University of Illinois at Urbana-Champaign
104 S. Mathews St.
Urbana, IL 61801
(217) 333-2346
deg@uiuc.edu

Abstract

There is a tradeoff between speed and accuracy in fitness evaluations when various discretization sizes are used to estimate the fitness. This paper introduces discretization scheduling, which varies the size of the discretization within the GA, and compares this method to using a constant discretization. It will be shown that when scheduling the discretization, less computation time is used without sacrificing solution quality. Fitness functions whose cost and accuracy vary because of discretization errors from numerical integration are considered, and the speedup achieved from using efficient discretizations is predicted and shown empirically.

1 Introduction

In recent years, advances in the field of genetic algorithms (GAs) have allowed an increasing number of complex real-world problems in optimization, search, scheduling, and machine learning to be solved with GAs. One obstacle to the use of GAs in commercial applications is the evaluation expense. As GAs typically require hundreds or thousands of function evaluations, and industrial applications may require anywhere from minutes to days for a single evaluation, a GA may require the good part of a year to complete. Some of these applications with expensive function evaluations include computational fluid dynamics, structural optimization, and environmental applications. In many cases, coarse-grained parameters can be used to decrease the computation time by introducing error in the fitness evaluations. This paper focuses on how these types of problems can run effectively and accurately while dealing with considerable amounts of error in the evaluation.

The goal of this paper is to establish efficient guidelines for GAs that use a finite discretization to approximate a continuous system. To achieve this goal, an understanding of the effects of evaluation error on GA performance must be obtained, and a model must be created that will both predict the accuracy and computational requirements in an environment with evaluation error present. Although the specific situations are somewhat idealized, the general idea applies to more complex evaluations resulting from implicit or explicit quadrature in finite elements, finite differences or other techniques used to approximate differential and integral equations.

This paper considers the following question: For multidimensional problems, how can the discretization be scheduled in order for the least amount of computation time to be spent toward finding a solution of a given quality? This paper is organized as follows. First, we go over background, including decomposition of GAs, a literature review and error analysis of numerical integration. Next, we introduce the concept of discretization and discuss its components: convergence time, population sizing, and discretization considerations. Naive and efficient discretizations are then defined. Two-dimensional experiments are run to show the computational savings. Finally, this paper is concluded.

2 Background

In this paper, it is assumed that in the design of competent GAs—GAs that solve hard problems quickly, reliably, and accurately—can be decomposed into subproblems, and that each subproblem can be analyzed separately. The principles of the sixfold decomposition of GAs (Goldberg, Deb, & Clark, 1992) are

1. Know what the GA processes: building blocks (BBs)

2. Ensure there is an adequate initial supply of BBs
3. Ensure that necessary BBs are expected to grow
4. Ensure that BB-decisions are well made
5. Solve problem of bounded BB-difficulty
6. Ensure that BBs can be properly mixed

We are particularly interested in the third and fourth items, ensuring that necessary building blocks are expected to grow and that building block decisions are well-made. Efficient techniques that introduce error to the fitness evaluation may not ensure that the necessary building blocks grow if the decisions are made poorly. That is, error introduced by efficient techniques cannot be so large that the GA cannot choose correctly between two distinct, competing individuals in most cases, or the necessary building blocks will be lost from the population.

2.1 Literature Review

Since using discretization in a fitness function approximates the true fitness, it is critical to understand the effects of error on GA performance. In this section, we review previous research efforts that analyze and explain the effects of error on GAs. Although the type of error that we analyze is due to discretization and searching on a grid, research efforts that investigate other types of error, such as sampling error, can help to shed light on general effects of error in GA performance.

Early studies on the effects of evaluation error on GA performance considered evaluation error that was mainly due to variance or randomness. These studies used sampling to estimate the fitness and ran the GA in a bounded computation time. The tradeoff for these problems is between the time to make each fitness evaluation and the total number of evaluations made. Either more, inaccurate evaluations or fewer, accurate evaluations are made. In addition, each implementation has different convergence time and population requirements. Optimal sampling in GAs was studied by (Fitzpatrick & Grefenstette, 1988; Miller & Goldberg, 1996; Aizawa & Wah, 1994).

Other types of research considered industrial applications of GAs, with expensive fitness evaluations. One approach was to create an approximate model to the fitness landscape with statistical techniques, such as neural networks (El-Beltagy, Nair, & Keane, 1999; Jin, Olhofer, & Sendhoff, 2000). However, many of these types of approaches are particularly prone to converging to a sub-optimal solution because they search in the space as defined by the statistical technique, not the original problem. Others considered heuristics to improve GA performance for certain types of problems (Le Riche & Haftka, 1993; von Wolfersdorf, Achermann, & Weigand, 1997; Kogiso, Watson, Gürdal, & Haftka, 1993), but these approaches are strictly empirical and cannot be applied to other types of problems. Finally, the injection island GA refined the model in a stepwise process by running multiple grids for large engineering applications in parallel (Lin, Punch, & Goodman, 1994; Punch, Averill, Goodman, Lin, Ding, & Yip, 1994). Intuitively, the authors understood that some good building blocks could be generated inexpensively with a coarse grid spacing, so they ran several different resolution grids in parallel, each of which was run on a different set of processors. They then injected the best individuals into the higher resolution grids.

Albert and Goldberg (2000) empirically examined the tradeoffs between more and less accurate models that use numerical integration. When using a constant grid with a deterministic fitness function, bias error is added to the evaluations. They found that for a bounded computation time, an optimal number of grid points exists that can maximize solution quality. This research provided the initial motivation for discretization scheduling because the solution quality with this method may not be that which is desired.

Albert and Goldberg (2001) examined the amount of solution quality limited by the computation time available under one-dimensional integrated fitness functions. The objective was to minimize the computation time such that a given solution quality is acquired. They varied the discretization by using fewer grid points in the early generations and exponentially increasing the number of grid points in order to more efficiently use computation time. Although more function evaluations are needed for the efficient method compared to when the number of grid points is constant for the entire GA, they achieved a significant computational speedup from their method.

2.2 Error Analysis

It is assumed that any numerical integration scheme used to estimate the fitness can be written of the form:

$$I(f) = \int_{a2}^{b2} \int_{a1}^{b1} f(x,y)dxdy \approx \sum_{i=1}^{n1} \sum_{j=1}^{n2} w_{ij} f(x_i, y_j)$$

where the integral of the function f taken at the points x_i and y_j is taken from *a1* to *b1* in the x-dimension and from *a2* to *b2* in the y-dimension, n_1 and n_2 are the total number of grid points in the x- and y-dimensions,

and w_{ij} are the weights as determined by the numerical integration scheme. The estimated error in the integration scheme is accurate to a certain order depending on h, the size of the discretization where h is $\frac{b1-a1}{n1}$ or $\frac{b2-a2}{n2}$. For the left-endpoint rule, the integral is $O(h)$ accurate; for the midpoint and trapezoidal rules, the integral is $O(h^2)$ accurate; and for Simpson's rule, the integral is $O(h^4)$ accurate.

In addition to the accuracy, there are different computational requirements to implement different numerical integration schemes. It is assumed that the time to estimate an integral is dependent only on the number of grid points, n. That is, it is assumed that there are no additional costs to calculate the weights w_{ij}.

3 Discretization Scheduling

This section focuses on discretization scheduling and its components, and particularly the effects of the error that is introduced when using a coarse grid. The topics critical to discretization scheduling are apparent variance error, convergence time and population sizing.

The concept of discretization scheduling is to use more than one discretization size within the GA. The choice of the number of grid points n is traditionally fixed within the GA. This choice of n is too precise at the beginning of the run, which leads to wasted computation cycles early on. A more efficient discretization would be to use fewer grid points in the first generations and increase the number of grid points throughout the GA. This is inspired by domino convergence (see section 3.2), which indicates that in the first few generations, the GA will be considering only the most salient bit. In theory, only two grid points are needed at the beginning of the run, but in practice, this number may need to be larger.

Since convergence time tells us which building blocks are converging at what time, how the grid should be spaced can be determined in order to efficiently converge to that particular building block. That is, if two grid points are initially used by the GA, the number of grid points must be doubled in order to converge to the second bit. Thus, increasing the number of grid points exponentially will match the salience of the converging bits when the bits are converted to a floating point number.

In the remainder of this section, we further elaborate on discretization scheduling. This section begins with the deriving of the computational requirements for GAs with fitness functions that consider numerical integration. However, much of the theory can be generalized to other, similar types of fitness functions.

The fitness function considered is an sum of m two-dimensional integrals. The experiments have the following form where f is the fitness function:

$$f = \sum_{i=1}^{m} \int_0^{x_i} \int_0^{y_i} \frac{\partial^2 g_i(x_i, y_i)}{\partial x_i \partial y_i} \partial y_i \partial x_i \qquad (1)$$

It is assumed that *g(x,y)* is continuous and differentiable.

3.1 Apparent Variance Error

When the discretization is scheduled—more than one grid spacing is used by the GA—apparent variance error is introduced into the GA. That is, although the evaluations are determinate, each time the grid changes, the fitness evaluation for a given set of variables also changes. When the grid changes often, the variance term in the error can become significant. In practice, this means that the GA will experience duration elongation—the GA will take more generations to converge—and will necessitate a larger population.

The ratio of the error variance to the fitness variance, r, at any time during the GA is assumed to be constant since both depend on the accuracy of the numerical scheme used as well as the distribution of individuals in the fitness landscape. It can be defined as

$$r = \frac{\sigma^2_{E,t}}{\sigma^2_{F,t}} \qquad (2)$$

where $\sigma^2_{E,t}$ is the error variance and $\sigma^2_{F,t}$ is the fitness variance, each at generation t. In addition, both the error and fitness variances decrease with time and are approximately zero at convergence. For the computational experiments in the following section, it is empirically shown that r is approximately constant during the GA when an efficient discretization is used.

3.2 Convergence Time

Convergence time is crucial for understanding how GAs perform and computation time can be used efficiently. In this section, the convergence time equations for exponentially-scaled building blocks are introduced, and the convergence time for a special case of a sum of identical subfunctions is displayed. The convergence times are based on selection intensity adapted from population genetics (Kimura, 1964).

Binary integer subcodes experience domino convergence, which converge starting with the most salient building block down to the least salient building block.

Since binary integer subcodes have exponentially-scaled bits, it is assumed that other exponentially-scaled problems also converge in this way. Thierens, Goldberg, and Pereira (1998) show that for selection operators with constant selection intensity, such as tournament selection, the population converges in $O(l)$ time where l is the string length. The convergence time for binary integer subcodes using pairwise tournament selection can be expressed as

$$t_{conv} = \frac{-\ln 2}{\ln\left[1 - \frac{I}{\sqrt{3}}\right]}\lambda = c_t \cdot k \tag{3}$$

where I is the selection intensity, k is the number of building blocks, and c_t is the time to converge to each successive building block (Thierens, Goldberg, & Pereira, 1998). The convergence time for the entire GA can be found when considering that $k = l$ in the above equation. This equation implies that it takes the same amount of time to converge to each building block, no matter what its fitness contribution. When the whole string converges, λ is equal to the string length l.

The convergence times are different when the fitness function is composed of a sum of identical subfunctions, when the fitness can be written as:

$$f = \sum_{i=1}^{m} g(x, y)$$

The convergence time are developed in (Albert, 2001) and the derivations are not printed here. When evaluation error is not present, the convergence time can be written as:

$$t_{conv1} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi}}\right]}k = c_t\sqrt{m}k \tag{4}$$

where $c_t \approx \ln 2\sqrt{3\pi} = 2.13$, and the total convergence time increases by a factor of $\sqrt{m}$. The convergence rate is $O(\sqrt{m}k)$ for these types of functions.

When evaluation error is present, as mentioned in section 3.1, the convergence time is elongated. The convergence time changes to

$$t_{conv2} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi(1+r)}}\right]}k = c_t\sqrt{m(1+r)} \cdot k \tag{5}$$

where, again, $c_t \approx \ln 2\sqrt{3\pi}$. The convergence rate is $O(\sqrt{m(1+r)}k)$ for these types of functions. This equation collapses to equation 4 when accurate fitness evaluations are used. It is assumed that r is constant or is approximately constant during the GA. It should be noted that it takes $c_k = c_t\sqrt{1+m}$ generations to converge to each bit in each subfunction.

The ratio of the convergence times can be determined from taking the ratio of equation 4 to equation 5. The ratio can thus be written as

$$\frac{t_{conv1}}{t_{conv2}} = \frac{c_t\sqrt{m}k}{c_t\sqrt{m(1+r)}k} = \frac{1}{\sqrt{1+r}} \tag{6}$$

Since $r \geq 0$, the above ration is always less than or equal to 1. That is, a GA with an accurate fitness function is on average expected to converge faster than its error-prone counterpart.

Because the least salient building blocks do not experience any selection pressure for a number of generations, it is possible that they will experience genetic drift and will converge randomly to either 0 or 1. The expected time for a bit to experience drift is proportional to the population size (Goldberg & Segrest, 1987):

$$t_{drift} \approx 1.4N \tag{7}$$

In this equation, N is the population size. For any application, it is important to ensure that drift will not occur until well after the population is expected to converge.

3.3 Population Sizing

Several population-sizing models have been derived for problems with equally salient building blocks, and more recently, population requirements have been considered for problems with exponentially-scaled building blocks. Initially, the population size for exponentially-scaled building blocks was observed empirically. When considering the drift model, it has been observed that the population size varies in the same way that the convergence time varies—linearly. That is, a single, exponentially-scaled variable is expected to converge as $O(l)$ and has also been empirically observed that the population size must vary as $O(l)$ as well (Lobo, Goldberg, & Pelikan, 2000).

Recently, Rothlauf (2001) developed a population-sizing model for problems with exponentially-scaled building blocks by considering the drift model. Drift only affects the GA if $t_{drift} < t_{conv}$, and if drift occurs, the least salient bits will randomly converge to either 0 or 1. If the bit string of length l is composed of m concatenated sub-strings of length k, each of which is exponentially-scaled, Rothauf found that the population size varies as $O(k\sqrt{m})$. However, the model by Rothauf does not take into account building-block mixing, and the resulting population sizes given by this model are inadequate for our purposes.

In section 3.2 , it was shown that a GA without evaluation error will converge as $O(k\sqrt{m})$. This implies that when evaluation error is present in the GA, the population size must vary as $O(k\sqrt{m(1+r)})$ because the GA converges at the slower rate of $O(k\sqrt{m(1+r)})$. From the observation of Lobo, Goldberg, and Pelikan (2000) and the theoretical work of Rothlauf (2001), it can be inferred that the convergence time and population size varies in the same way for exponentially-scaled problems.

The population sizes for our experiments are written in the form

$$N_n = c_N\sqrt{m}k$$
$$N_e = c_N\sqrt{m(1+r)}k$$

where N_n is the naive population size and N_e is the efficient size. The constant c_N is the same in both equations and can be found empirically when using a naive discretization. The population size when using the efficient discretization can be determined once r is known.

3.4 Putting it All Together

A naive discretization assumes that the discretization is constant throughout the duration of the GA. This can be contrasted with an efficient discretization, in which the discretization varies within the GA (Albert, 2001). The computation time for a GA with a multiple integral can be modeled as

$$T = (\alpha + \beta n_1 n_2 ... n_d)GN_n.$$

where α is the overhead per individual per generation, β is the time to calculate one sample, n_i is the number of samples in the ith dimension, $G = t_{conv}$ is the total number of generations to convergence, and N is the population size. The term $(\alpha + \beta n_1 n_2 ... n_d)$ is the cost of each function evaluation. When the number of grid points in any dimension is identical, and $t_{conv} = c_t\sqrt{m}k$ is substituted for G, the above equation can be simplified to

$$T_n = (\alpha + \beta n^d)c_t k\sqrt{m} \cdot N_n. \quad (8)$$

where d is the number of dimensions, and N_n is the naive population size.

It is assumed that the discretization is scheduled in multiple dimensions by starting with two grid points in each dimension. Since the convergence time models tells us which bit the GA is converging to, it is also known how many grid points are used in each dimension. That is, when the GA is converging to the first bit, $2^1 = 2$ grid points are needed to converge to the correct first bit. It will take c_k generations to converge to the first bit, and the GA will start to converge to the second bit, when it will require $2^2 = 4$ grid points in each dimension. The GA will proceed in this way until it converges to the final kth bit when 2^k grid points are needed. Hence, the number of grid points in each dimension must be doubled in every dimension every c_k generations.

The assumption that only two discretizations are needed in each dimension may be insufficient for the GA to converge to the correct optima, so a larger number of grid points may be needed to start with. It is assumed that at any given time, the number of grid points in each dimension is the same, but an efficient time budget can be easily derived when the number of grid points varies in every direction.

For an arbitrary amount of dimensions, d, the efficient time, as described above, can be written as

$$T_e = \sum_{i=1}^{k}(\alpha + \beta \cdot 2^{d\cdot i})c_k N_e.$$

The above equation can be rewritten after substituting $N_e = N_n\sqrt{1+r}$ and $c_k = c_t\sqrt{m(1+r)}$:

$$T_e = \sum_{i=1}^{k}(\alpha + \beta \cdot 2^{d\cdot i})c_t\sqrt{m}N_n(1+r). \quad (9)$$

The speedup can be written in terms of the naive and efficient computation times:

$$S = \frac{T_n}{T_e} = \frac{(\alpha + \beta n^d)c_t\sqrt{m}k \cdot N_n}{\sum_{i=1}^{k}(\alpha + \beta \cdot 2^{d\cdot i})c_t\sqrt{m} \cdot N_n(1+r)} \quad (10)$$

Equation 10 can be simplified and rewritten knowing that $\sum_{i=1}^{k} 2^i = 2^{k+1} - 2$ and $n = 2^k$ as

$$S = \frac{\frac{\alpha}{\beta}k + k2^{k\cdot d}}{\frac{\alpha}{\beta}k + (2^{k+1}-2)^d} \cdot \frac{1}{1+r}.$$

4 Experiments

The fitness function considered is an integral in two dimensions. The experiments have the following form where f is the fitness function:

$$f = \sum_{i=1}^{3}\int_0^{x_i}\int_0^{y_i}\frac{\partial^2 g_i(x_i,y_i)}{\partial x_i \partial y_i}\partial y_i \partial x_i \quad (11)$$

$$g(x,y) = (e^{Ax}\cos(Bx))(e^{Ay}\cos(By)) \quad (12)$$

The fitness function is the sum of three identical subfunctions, where A and B are 0.1 and 1.4, respectively.

Six bits are used to represent the string for each x and y variables when the variables are mapped from 0 to 10.08. Thus, the total chromosome length is 36 bits. The maximum occurs at 6.88 for each x and each y in each subfunction.

The integration ideally should start with two grid points in each direction. However, two grid points are inadequate for the integration because the function is multimodal. An analysis shows that at least 8 grid points are needed in each direction for the GA to choose the correct first bit. When fewer grid points are used, the GA starts to converge to a local optimum and correct bits are lost. This number is found by starting with 2 grid points and determining the largest difference between the location of the approximated optima and the actual optima. If this value is larger than the discretization size, h, then the number of grid points in increased. For this problem, the smallest discretization size such that the location of the approximated optima is within h of the true optima is 5, and 8 is the smallest power of two greater than 5.

4.1 Apparent error variance

A minimum of 8 grid points is needed for the GA to find the correct solution. When switching from 8 to 16 grid points, apparent variance error is added to the fitness evaluations. The value of the variance is approximately 0.80 and it is roughly constant during the GA. Figure 1 shows how r varies within the GA. This value of r is used to predict the convergence time and the population size.

4.2 Convergence time analysis

The convergence time equation must be modified to take into account the three subfunctions and two variables in each subfunction. For any given fitness function (Muhlenbein & Schlierkamp-Voosen, 1993) give the following equation to describe how the fitness mean varies from one generation to another:

$$\mu_{t+1} - \mu = \sigma_{F,t} I \tag{13}$$

and from (Albert & Goldberg, 2001), when the fitness function is a sum of m identical subfunctions, the above equation can be rewritten as

$$m(\mu_{t+1} - \mu) = \sqrt{(}\sigma_{F,t} I)$$

If the subfunction can be written as $g(x, y) = h(x) \cdot h(y)$, then the equation 13 is valid as written for $h(x)$ and $h(y)$, where $h(x)$ and $h(y)$ are identical when $x = y$. In the same way, $m' = m^2$, which adjusts for $g(x, y)$. Since m is 3, m' is 9:

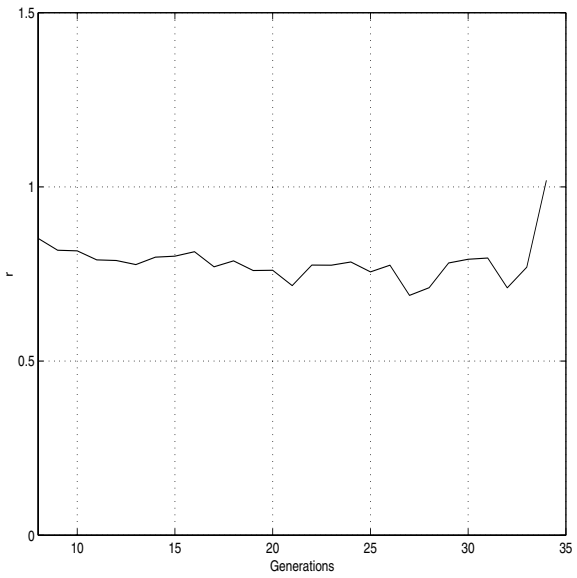


Figure 1: The value of r for the idealized two-dimensional experiments is roughly constant during the GA, and its value centers at 0.80

.

$$m'(\mu_{t+1}(\lambda) - \mu_t(\lambda)) = \sqrt{m'}\sigma_t(\lambda) I \tag{14}$$

The value $m' = 9$ is used in the convergence and population sizing equations. If binary tournament selection is considered, then the convergence times are

$$t_{conv1} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m'\pi}}\right]} k = c_t \sqrt{m'} k$$

$$t_{conv2} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m'\pi(1+r)}}\right]} k = c_t \sqrt{m'(1+r)} \cdot k$$

Using these values, the naive convergence time is 36.2 generations. Knowing that r is 0.80, the efficient convergence time is expected to be 49.2 generations.

4.3 GA parameters

The population size was empirically determined by the following equations:

$$N_n = c_N k \sqrt{m'}$$

$$N_e = c_N k \sqrt{m'(1+r)}$$

Experiments indicate that $c_N = 8.89$ for the population to be adequately sized, which results in a population size of 160 individuals for the naive experiments.

The population size for the efficient experiments is 215 individuals when $r = 0.80$. The drift time for the naive and efficient experiments are 224 and 301 generations, much longer than the predicted convergence times of 36.2 and 49.2 generations. Thus, drift is not a concern for these experiments.

For the experiments, the probabilities of selection and crossover are 1.0, and the probability of mutation is $1/N$ for this problem. Pairwise tournament selection, one-point crossover and uniform mutation are the operators used for this problem.

4.4 Time budgeting and speedup

The expected computation time for the naive case can be found when α and β are known. Here, the two dimensional case is considered, $d = 2$. Computational experiments indicate that α and β are 2.51×10^{-5} and 1.37×10^{-7}, respectively, when Mflops are used to estimate the computation time. To be able to discriminate between the least salient bit in each of the subfunction, 64 grid points are needed in each dimension. Equation 8 can predict the time needed for the naive implementation and the following equation predicts the time for the efficient implementation.

The naive computation time is 3.40 Mflops and the number of function evaluations necessary is predicted to be 5790. For the efficient case, 1.59 Mflops are needed and predicted number of function evaluations is 10,580. This yields a predicted speedup of 2.14. As mentioned previously, the GA is started with 8 grid points in each dimension. In other words, the first $3 \cdot 8.20 = 24.6$ generations are run with 8 grid points in both the x and y directions.

4.5 Results

For the experiments, 50 trials were run using both the naive and efficient discretization. The expected speedup from using an efficient discretization is 2.14. Tables 1 and 2 show the generations, time, and number of fitness evaluations for the naive and efficient implementations.

Table 1: Predicted and actual computation time values for two-dimensional experiments using a naive discretization

	Gens	**Time**	**Evaluations**
Predicted:	36.2	3.40	5790
Actual:	37.1	3.48	5940

As in the one-dimensional case, the efficient GA takes slightly longer to converge—50.0 generations compared to the 49.2 predicted generations. The actual speedup of 2.06 is very close to the predicted speedup of 2.14. That is, despite the efficient GA making almost twice as many function evaluations as the naive GA, on average, it finished in less than half the time. Figure 2 shows the converged gene position for each subfunction for the naive and efficient implementation. Like in the one-dimensional experiments, the experimental gene position follows very closely to the predicted gene position.

Table 2: Predicted and actual computation time values for two-dimensional experiments using an efficient discretization

	Gens	**Time**	**Evaluations**
Predicted:	49.2	1.59	10,750
Actual:	50.0	1.69	10,750

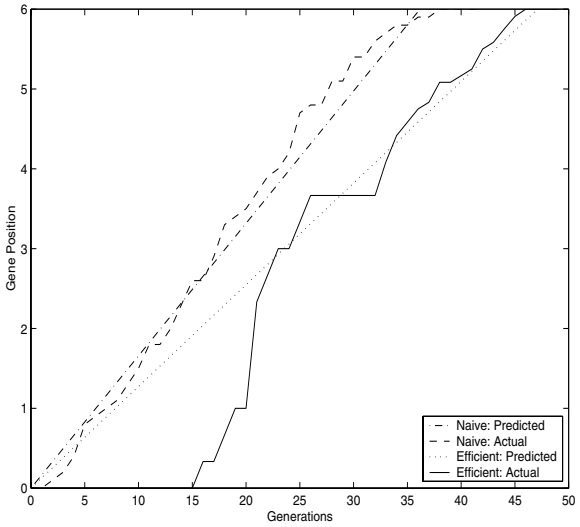


Figure 2: Predicted and actual gene position for idealized two-dimensional experiments

5 Conclusions

Discretization scheduling allows a GA to efficiently use computation time on a single processor by changing the grid spacing during the GA. This is particularly useful for fitness functions that use a discrete system to approximate a continuous system and that are computationally expensive. We show that, theoretically, it is possible to schedule the grid in multiple dimensions such that computation time is decreased while solution quality is not sacrificed. Empirically, we show that these relationships hold for a two-dimension nu-

merical integration problem.

Acknowledgments

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

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

References

Aizawa, A., & Wah, B. (1994). Scheduling of genetic algorithms in a noisy enviroment. *Evolutionary Computation*, *2*(2), 97–122.

Albert, L., & Goldberg, D. (2000). The effect of numerical integration on the solution quality of a genetic algorithm. *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, 15–21.

Albert, L., & Goldberg, D. (2001). Efficient evaluation relaxation under integrated fitness functions. *Intelligent Engineering Systems through Artificial Neural Networks*, 165–170.

Albert, L. A. (2001, December). *Efficient genetic algorithms using discretization scheduling.* Master's thesis, University of Illinois, Urbana-Champaign.

El-Beltagy, M., Nair, P., & Keane, A. (1999). Metamodeling techniques for evolutionary optimization of expensive problems: Promises and limitations. *Proceedings of Genetic and Evolutionary Computation Conference*, 196–203.

Fitzpatrick, J., & Grefenstette, J. (1988). Genetic algorithms in noisy environments. *Machine Learning*, *3*, 101–120.

Goldberg, D., Deb, K., & Clark, J. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D., & Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. *Proceedings of the Second International Conference On Genetic Algorithms*, 1–8.

Jin, Y., Olhofer, M., & Sendhoff, B. (2000). On evolutionary optimization with approximate fitness functions. *Proceedings of the Genetic and Evolutionary Computation Conference*, 786–793.

Kimura, M. (1964). *Diffusion models in population genetics*, Volume 2 of *Supplementary review series in applied probability.* London: Methuen.

Kogiso, N., Watson, L. T., Gürdal, Z., & Haftka, R. T. (1993). Genetic algorithms with local improvement for composite laminate design. *Structures and Controls Optimization*, *38*, 13–28.

Le Riche, R., & Haftka, R. T. (1993). Optimization of laminate stacking for buckling load maximization by genetic algorihm. *AIAA Journal*, *31*(5), 951–956.

Lin, S. C., Punch, W. F., & Goodman, E. D. (1994). Coarse-grain parallel genetic algorithms: Categorization and new approach. *Sixth IEEE Parallel Distributed Processing*, 28–37.

Lobo, F. G., Goldberg, D. E., & Pelikan, M. (2000). Time complexity of genetic algorithms on exponentially scaled problems. *Proceedings of the Genetic and Evolutionary Computation Conference*, 151–158.

Miller, B., & Goldberg, D. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131.

Muhlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. continous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Punch, W. F., Averill, R. C., Goodman, E. D., Lin, S. C., Ding, Y., & Yip, Y. C. (1994). Optimal design of laminated composite structures using coarse-grain parallel genetic algorithms. *Computing Systems in Engineering*, *5*(4-6), 415–423.

Rothlauf, F. (2001). *Towards a theory of representations for genetic and evolutionary algorithms: Development of basic concepts and its application to binary and tree representations.* Doctoral dissertation, University of Bayreuth, Germany.

Thierens, D., Goldberg, D. E., & Pereira, A. G. (1998). Domino convergence, drift and the temporal-salience structure of problems. *The 1998 IEEE Conference on Evolutionary Computation Proceedings*, 535–540.

von Wolfersdorf, J., Achermann, E., & Weigand, B. (1997). Shape optimization of cooling channels using genetic algorithms. *Transactions of the ASME*, *119*, 380–386.

Eugenic Evolution Utilizing a Domain Model

Matthew Alden
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
malden@cs.utexas.edu

Aard-Jan van Kesteren
Department of Computer Science
University of Twente
Enschede, The Netherlands
ajvankesteren@hotmail.com

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
risto@cs.utexas.edu

Abstract

In this paper we introduce The Eugenic Algorithm with Modeling (TEAM), an evolutionary search algorithm that employs statistical analysis to promote construction of high-fitness chromosomes. A model of gene/fitness correlations is automatically generated to direct the construction process. When applied to the combinatorial optimization problems of finding a maximally weighted cut in a graph and minimizing the two-dimensional Rosenbrock function, TEAM performs well compared to other evolutionary algorithms at evolving high-fitness solutions.

1 INTRODUCTION

The field of combinatorial optimization focuses on problems with a finite number of possible solutions. For many such problems, an optimal solution cannot be found analytically, or the problem is too large for exhaustive search. This class of interesting problems includes the traveling salesman problem, maximally weighted cut in a graph, integer programs, the subset sum problem, and maximal clique in a graph. Near-optimal solutions can often be found reasonably fast using techniques such as hill-climbing and simulated annealing (SA; Kirkpatrick and Sherrington, 1988). These standard techniques operate by incrementally improving suboptimal solutions. Evolutionary algorithms, which utilize optimization strategies modeled after biological evolution, implement a more global search and have been shown to be particularly powerful on combinatorial optimization problems.

Numerous evolutionary algorithms have been designed to operate on a population of binary chromosomes, a convenient structure for encoding solutions to combinatorial problems. The traditional approach, exemplified by genetic algorithms (GAs; Holland, 1975), produces new chromosomes via recombination of existing chromosomes, with a component of mutation. The information inherited by a single new chromosome is derived from only a small percentage of the total information present in the population.

Recently, evolutionary algorithm research has progressed towards increasingly constructive techniques for generating new chromosomes. Among these algorithms are Binary Simulated Crossover (BSC; Syswerda, 1993), Population Based Iterative Learning (PBIL; Baluja, 1994), and the Eugenic Algorithm (EuA; Prior, 1998). These algorithms construct chromosomes based on information in the entire chromosome population. A probability indicating allele preference is calculated for every gene, and these probabilities are then used to bias the allele selection process towards an estimated ideal chromosome.

This paper presents a second-generation constructive algorithm, The Eugenic Algorithm with Modeling (TEAM), that applies statistical methods to the identification of desirable alleles. A model of gene/fitness correlations is built and used during chromosome construction to aid allele selection.

We next describe the TEAM algorithm. In Section 3 we present and discuss the results of applying TEAM to two combinatorial optimization problems, finding a maximally weighted cut in a graph and minimizing the two-dimensional Rosenbrock function (De Jong, 1975), showing that TEAM performs better than stochastic hillclimber, standard GA, and EuA. Finally, we speculate how the algorithm might be improved by using different statistical tests and generalizing the model.

2 TEAM

2.1 OVERVIEW

At a high level, TEAM operates similar to other GAs. A population of chromosomes is evolved based on feedback from a chosen chromosome evaluation function, $f(c)$. Augmenting the standard chromosome population, TEAM maintains a gene/fitness correlation model, hereafter referred to as the *model*, and several sets of fitness values used in updating the model, hereafter referred to as the *model statistics*. For every gene, these additional structures are used to estimate the allele with the greatest likelihood of producing a high-fitness chromosome.

The algorithm begins by initializing three data sets as follows: (1) build the initial population, (2) initialize the model statistics based on the initial population, (3) initialize the model based on the model statistics. Population evolution proceeds via repetition of the following steps: (1) create a new chromosome, (2) select and replace an existing chromosome in the population with the new chromosome, (3) update the model statistics using the new chromosome, (4) reconstruct the model if necessary. Evolution terminates after a specified number of generations has elapsed.

Chromosome creation involves assigning an allele to every gene in the genome. The first genes assigned are those with the strongest observed influence on fitness. The allele with the strongest observed correlation to high fitness is assigned to each gene. Allele/fitness correlation is partially based on previous allele assignments, therefore a partially constructed chromosome contributes to its own construction.

After the entire chromosome has been constructed, a chromosome in the existing population is selected for extinction by one of several heuristics. The selected chromosome is removed from the population, and the newly constructed chromosome added. The model statistics are updated with information about the new chromosome. The model is updated periodically, after a specified number of elapsed generations since the previous update.

2.2 CHROMOSOME CREATION

Chromosome creation involves ordering the genes and then assigning an allele to every gene. The assignment of an allele x to gene g_i is called a *binding*, written $g_i = x$. For binary chromosomes, $x \in \{0,1\}$. A chromosome c of length l is defined by a set of bindings, $\{c_{g_1}, c_{g_2}, \ldots, c_{g_l}\}$, where c_{g_i} is the allele of gene g_i in chromosome c. A chromosome c is said to satisfy a set of bindings S if for every binding $g_i = x$ in S, $c_{g_i} = x$.

The first step of chromosome creation, gene ordering, is based on *selectivity*. The selectivity of a gene is an estimation of how clearly the observed fitness values in the population suggest a particular allele for that gene. For example, if the set of fitness values of chromosomes satisfying binding $g_i = 0$ is $\{12, 15, 16, 17\}$, and the set of fitness values of chromosomes satisfying binding $g_i = 1$ is $\{18, 19, 20, 23\}$, then the gene has high selectivity; the allele 1 appears to consistently lead to higher fitness chromosomes. Alternatively, if the respective sets of fitness values were $\{12, 15, 19, 20\}$ and $\{16, 17, 18, 23\}$, the gene's selectivity would be low. Genes are ordered by decreasing selectivity. The selectivity of a gene g_i relative to population P is estimated by:

$$Sel(g_i, P) = \begin{cases} 1.0, \text{if } \left|[P]^F_{g_i=0}\right| \leq 1 \vee \left|[P]^F_{g_i=1}\right| \leq 1 \\ t\_test\left([P]^F_{g_i=0}, [P]^F_{g_i=1}\right), \text{otherwise} \end{cases},$$

where $[P]^F_{g_i=x}$ is the set of fitness values of chromosomes from population P that satisfy binding $g_i = x$. Function t_test returns one minus the observed significance level (OSL) used in Student's t-test (Press et al., 1992), a statistical hypothesis test indicating whether two sample sets are believed to come from the same normal distribution. A high value for t_test, near 1, indicates that the sets are probably not from the same distribution, while a low value, near 0, indicates that there is insufficient evidence to make that distinction.

Once the genes have been ordered, they are sequentially bound to alleles in this order. Set B, initially empty, contains the bindings that define the partially constructed chromosome. As each gene is bound, the resulting binding is added to set B.

To select an allele for gene g_i, a set B'_i is first calculated. B'_i is a subset of B containing the bindings deemed most relevant to the pending binding of gene g_i. Construction of B'_i is described in Section 2.3. An estimate of the allele, x_i, for gene g_i most likely to lead to a high-fitness chromosome is calculated from B'_i as:

$$x_i = \begin{cases} 0, \text{if } \overline{[P]^F_{B'_i \cup g_i=0}} > \overline{[P]^F_{B'_i \cup g_i=1}} \\ 1, \text{otherwise} \end{cases},$$

where $[P]_{B'_i \cup g_i=y}$ is the subpopulation of P composed of chromosomes satisfying all bindings in the set $B'_i \cup g_i = y$, and $[P]^F_{B'_i \cup g_i=y}$ is the set of fitness values of chromosomes in that subpopulation. The gene g_i is bound to allele x_i with probability $1 - \frac{1}{2}(1 - Sel(g_i, [P]_{B'_i}))^\alpha$, otherwise gene g_i is bound to allele $1 - x_i$. Parameter α regulates the dependence of allele selection on selectivity. This probabilistic scheme assigns alleles based on confidence in

expected outcome. Statistically influential genes are more likely to be assigned x_i, while genes not appearing to strongly influence fitness are assigned more randomly. In this way, the system performs neighborhood search to fine tune less influential genes. As the last step, to promote diversity, the binding of gene g_i is mutated with probability p_m.

2.3 THE MODEL

The model's primary purpose is provide a basis for estimating gene epistatis (Davidor, 1991). The model records the observed relative influence of genes on chromosome fitness in a set of gene rankings. Each ranking describes the influence of a single gene on fitness when used in conjunction with other genes. For example, a potential model for a population of chromosomes with four genes might look like the following:

$$\begin{array}{llll} g_0: & g_3 & g_1 & g_2 \\ g_1: & g_2 & g_0 & g_3 \\ g_2: & g_0 & g_1 & g_3 \\ g_3: & g_2 & g_0 & g_1 \,. \end{array}$$

For each gene, all other genes are sorted based on the quantity $U_r(g_i, g_j)$, an estimation of the amount of influence the combination of genes g_i and g_j has on fitness. An ordered pair of genes, (g_i, g_j), is considered highly influential if for $g_j = x$, the set of estimated fitness values of chromosomes satisfying the bindings $\{g_i = 0, g_j = x\}$ is statistically different from the set of estimated fitness values of chromosomes satisfying $\{g_i = 1, g_j = x\}$. A high U_r value, near 1, indicates high influence, and a low value, near 0, indicates low influence. If $U_r(g_i, g_x) > U_r(g_i, g_y)$ then gene g_x appears before gene g_y in the entry for gene g_i. $U_r(g_i, g_j)$ is calculated as:

$$U_r(g_i, g_j) = \frac{1}{4} \begin{pmatrix} t\_test(F_{g_i=0}, F_{g_i=0,g_j=0}) & + \\ t\_test(F_{g_i=0}, F_{g_i=0,g_j=1}) & + \\ t\_test(F_{g_i=1}, F_{g_i=1,g_j=0}) & + \\ t\_test(F_{g_i=1}, F_{g_i=1,g_j=1}) & \end{pmatrix},$$

The sets F comprise the model statistics. $F_{g_i=x}$ is the set of fitness values of all previously evaluated chromosomes that satisfy binding $g_i = x$. For a chromosome of length l, there are $2l$ possible single bindings, and therefore $2l$ $F_{g_i=x}$ sets. $F_{g_i=x,g_j=y}$ is the set of fitness values of all evaluated chromosomes which satisfy bindings $\{g_i = x, g_j = y\}$. There are $4l^2$ such $F_{g_i=x,g_j=y}$ sets, though all sets such that $i = j$ are unused. These sets F act as a global history; as evolution proceeds the fitness value of every evaluated chromosome is a member of l of the $F_{g_i=x}$ sets and $l(l-1)$ of the $F_{g_i=x,g_j=y}$ sets. $\overline{F_{g_i=x}}$ and $\overline{F_{g_i=x,g_j=y}}$ are estimations of the fitness value of any chromosome satisfying a specified single or double binding. Although these sets are incomplete, we assert that they eventually contain sufficient information to support $U_r(g_i, g_j)$.

During allele selection, a set B'_i is calculated from set B, the set of bindings comprising a partial chromosome. Let $h_{g_i,j}$ be the j^{th} gene in the model entry for g_i. B'_i is initially empty, and for each gene $h_{g_i,j}$ in the model entry for g_i, proceeding via increasing j, we perform the following test: if $h_{g_i,j}$ is part of a binding $b \in B$ and $\left|[P]_{B'_i \cup b}\right| \geq n_{min}$ then add b to B'_i. Parameter n_{min} specifies the minimum number of chromosomes that must be present in $[P]_{B'_i}$.

2.4 REPLACEMENT POLICY

After a new chromosome has been created, an existing chromosome is selected for removal from the population. A typical heuristic for extinction is poor fitness, i.e. the least fit chromosome is removed. However, based on the information in the model we can make a more intelligent selection. We can remove the chromosome that contributed the least to the construction of the new chromosome.

In the course of creating a new chromosome, the sets B'_i are calculated l times. The more times a chromosome appears in instances of $[P]_{B'_i}$, the more that chromosome has contributed to the construction of the new chromosome. Therefore, a reasonable extinction heuristic is to remove the chromosome that appeared least often in instances of $[P]_{B'_i}$. This heuristic regards chromosomes as informational units rather than simple patterns for high fitness, promoting the retention of information in the population.

These policies are called *replace_worst_fitness* and *replace_worst_contributor*, and they will be tested experimentally below.

3 EXPERIMENTS

TEAM was evaluated on two combinatorial optimization problems and compared to two standard algorithms as well as EuA, the predecessor of TEAM.

3.1 ALGORITHMS

3.1.1 Stochastic Hillclimber

A simple Stochastic Hillclimber was used as a strawman in these experiments. The hillclimber maintains one chromosome. Each generation, a new chromosome

is created solely by mutating the existing chromosome. The probability of each bit being mutated is specified by parameter p_m. If the fitness of the new chromosome is higher than that of the original chromosome, the original chromosome is replaced by the new chromosome. Evolution continues until a specified number of generations has elapsed.

3.1.2 Genetic Algorithm

The GENEsYs-1.0 (Bäck, 1992) GA implementation was used in these experiments. Fitness-proportionate selection, elitism, and 2-point crossover were used. The GA is further parameterized by the population size, n, the per-chromosome probability of recombination, p_r, and the per-gene probability of mutation, p_m.

3.1.3 EuA

The predecessor of TEAM, EuA uses statistical predictors of chromosome fitness without maintaining a model or model statistics. A detailed description of this algorithm can be found in (Prior, 1998) and (Polani and Miikkulainen, 2000). This algorithm is parameterized by the population size, n, the probability of allele selection noise (similar to mutation), p_n, and the per-gene probability that an extinct allele will be reintroduced (only applicable when all chromosomes have the same allele for a particular gene), p_i.

3.1.4 TEAM

As described in Section 2, TEAM is parameterized by the population size, n, the minimum number of chromosomes to consider during allele selection, n_{min}, the per-gene probability of mutation, p_m, the selectivity factor, α, the model update frequency, T_{model}, and the chromosome replacement policy, ρ.

3.2 RESULTS

Our two combinatorial optimization test problems differ in how amenable they are to neighborhood search algorithms. Finding a maximally weighted cut in a graph, though an NP-complete problem, has a high degree of hillclimbability, making neighborhood searches effective. The Rosenbrock Function, on the other hand, has many local optima in binary space (Prior, 1998). This problem is very deceptive, making neighborhood searches less effective. The algorithms in this paper direct and focus neighborhood search differently, therefore these problems will illuminate the algorithms' strengths and weaknesses.

3.2.1 Maximally Weighted Cut in a Graph

In this problem we wish to partition the vertices of an undirected, weighted graph into two sets, V_0 and V_1, such that the sum of weights of all edges having one endpoint in V_0 and the other endpoint in V_1 is maximal. A feasible solution to this problem is a partition of the vertices such that every vertex is a member of V_0 or V_1, but not both. Given a graph of n vertices and m edges, we encode a partition as a binary chromosome of length n. If $g_i = 0$, then vertex i is a member of V_0, otherwise $g_i = 1$ and vertex i is a member of V_1.

The sum of weights of all edges crossing the partition is used as the chromosome evaluation function $f(c)$. It is calculated as

$$f(c) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} \left(c_{g_i} \left(1 - c_{g_j}\right) + c_{g_j} \left(1 - c_{g_i}\right) \right),$$

where w_{ij} is the weight of the edge incident on vertices i and j, or valued 0 if no such edge exists. The GENEsYs GA package implicitly minimizes fitness value, therefore an altered chromosome evaluation function $-f(c)$ was used with that package. The graph was constructed randomly, with edge weights uniformly distributed on $[0, 1]$.

Each algorithm was executed 100 times, each evolved for 100,000 generations. The average best fitness value of each generation is shown in Figure 1. The parameters used for each algorithm are: Stochastic Hillclimber: p_m=0.04; GA: n=50, p_r=0.6, p_m=0.001; EuA: n=100, p_n=0.05, p_i=0.01; and TEAM: n=100, n_{min}=20, p_m=0.01, α=0.6, T_{model}=100, ρ=*replace_worst_contributor*. These parameter values were determined experimentally; small variations produce roughly equivalent results.

All algorithms tested eventually generated chromosomes of approximately the same fitness. In this domain of few local optima, only the curves prior to the plateau are noticeably different. The hillclimber makes good progress in early generations, but EuA and TEAM soon catch up and exceed it in performance. The differences between the final average best fitness values of the four algorithms, though relatively small, are statistically significant.

Both of TEAM's chromosome replacement policies, *replace_worst_fitness* and *replace_worst_contributor*, were tested on this problem. The *replace_worst_contributor* policy (shown) generated chromosomes with fitness values approximately 10% better than those under the *replace_worst_fitness* policy. This result suggests that low-fitness individuals can contain valuable information, and are not always the best choice for extinction.

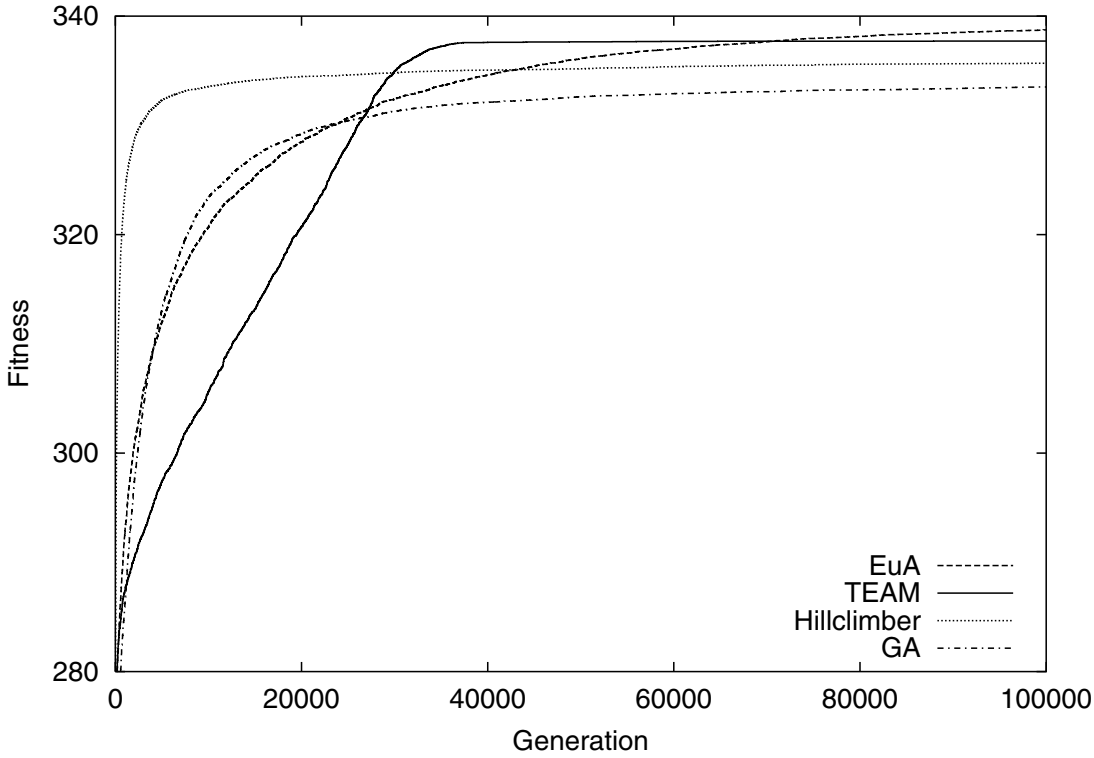


Figure 1: Average Best Fitness of 100 Runs Maximizing Cut Weight Over 100,000 Generations. In this problem with relatively few local maxima, the hillclimber makes good progress early on, but is eventually exceeded by EuA and TEAM. The final fitness differences are statistically significant ($p < 0.05$).

3.2.2 Two-Dimensional Rosenbrock Function

In this problem we wish to minimize the following function:

$$f(x, y) = 100(y - x^2)^2 + (x - 1)^2,$$

where $x, y \in [-5.12, 5.12]$. A feasible solution to this problem is a two-dimensional point, (x, y), which we encode as two floating-point numbers in a chromosome of 64 genes. We calculate the coordinates x and y as follows:

$$x = 5.12\left(1 - 2\sum_{i=1}^{32} 0.5^i c_{g_i}\right),$$

$$y = 5.12\left(1 - 2\sum_{i=33}^{64} 0.5^{i-32} c_{g_i}\right).$$

The coordinate x is encoded in genes 1 through 32, with gene 1 the most significant bit (MSB) in the floating-point encoding of x, and gene 32 the least significant bit (LSB). Similarly, coordinate y is encoded in genes 33 through 64, with gene 33 the MSB and gene 64 the LSB.

All algorithms used a chromosome evaluation function based directly on the Rosenbrock function value, $f(c) = f(x, y)$. 100 runs were performed for each algorithm, each evolved for 50,000 generations. The average best fitness value of each generation is shown in Figure 2. The parameters used for each algorithm are: Stochastic Hillclimber: p_m=0.3; GA: n=50, p_r=0.6, p_m=0.016; EuA: n=100, p_n=0.1, p_i=0.01; and TEAM: n=100, n_{min}=20, p_m=0.01, α=0.2, T_{model}=100, ρ=*replace_worst_fitness*. These parameter values were again determined experimentally, and small variations produce roughly equivalent results.

In this problem, TEAM clearly outperformed all other algorithms: it generated solutions with fitness values several orders of magnitude closer to optimal than its competitors. The differences between the final average best fitness of the four algorithms are statistically significant. The discretization of the problem introduces many local optima, traditionally difficult features for optimization algorithms. Yet during later generations, TEAM's average best fitness continues to improve, suggesting TEAM could yield even better solutions if evolution continued beyond 50,000 generations.

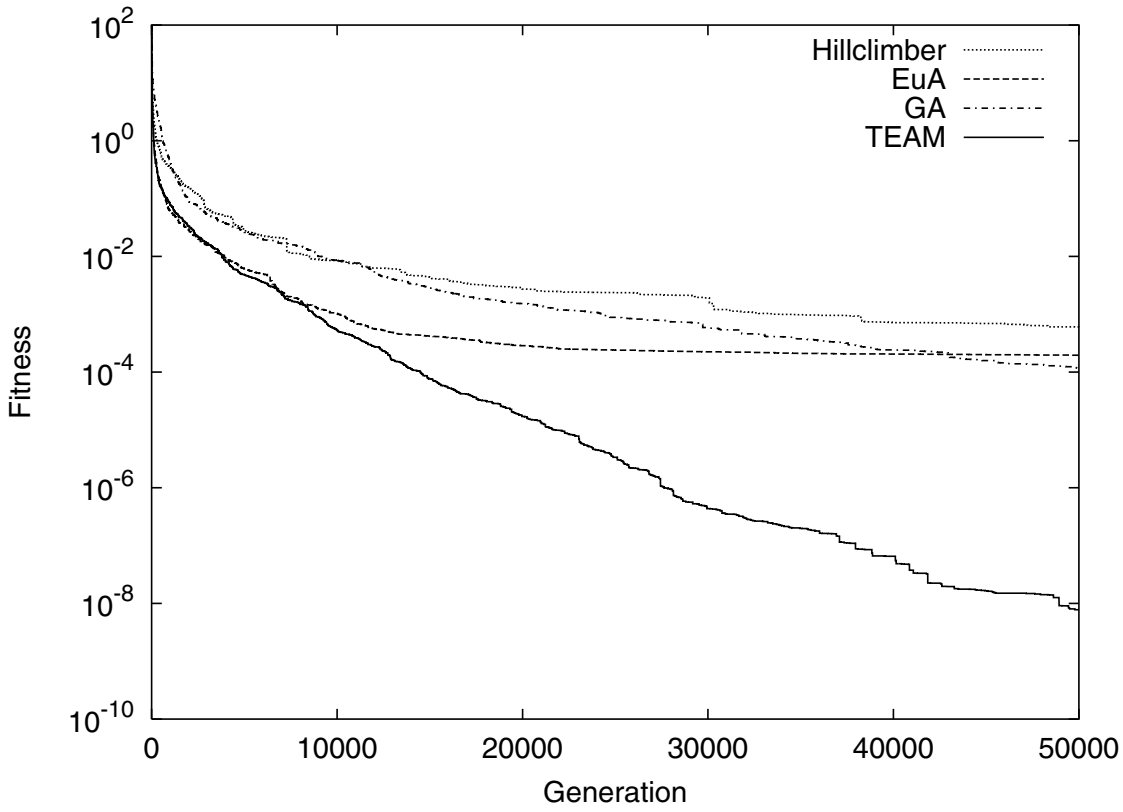


Figure 2: Average Best Fitness of 100 Runs Minimizing the Rosenbrock Function Over 50,000 Generations. In this problem with many local optima and significant deception, TEAM finds solutions several orders of magnitude better than the other algorithms. The final fitness differences are statistically significant ($p < 0.05$).

Both of TEAM's chromosome replacement policies were tried on this problem as well. Interestingly, the alternative *replace_worst_contributor* policy caused TEAM's population to often converge to suboptimal solutions. The more common *replace_worst_fitness* policy (shown) performed significantly better, allowing TEAM to consistently find good solutions.

How does TEAM achieve such a strong performance on this problem? To illustrate, recall that the order in which genes are bound during chromosome creation is determined by selectivity. Gene ordering therefore tells us which genes TEAM identified as the most useful, thereby illustrating its progress toward solution. Figure 3 shows such a histogram of the gene order. The gray scale indicates a gene's average rank during chromosome creation over several generations. Dark coloration indicates that the gene is among the first genes bound, and lighter coloration indicates that the gene is bound later.

The most prominent feature of Figure 3 is the concentrated dark bands, i.e. adjacent genes that were bound early in the chromosome creation process. In early generations, the two bands are localized near genes 1 and 2, at the bottom of the histogram, and genes 33 and 34, near the middle of the histogram. Recall that in the floating point encoding of (x, y) in a chromosome, these genes correspond to the most significant bits of the x and y coordinates. Since the two-dimensional Rosenbrock function has a relatively small region of near optimal solutions, this result makes sense: the most important indicators of fitness, and thus the most important genes to initially bind, are those responsible for the largest changes in the point (x, y). It is important to set those genes correctly, otherwise there is no possibility of high fitness.

Gene 33 is identified as selective almost immediately. All points in the domain with fitness relatively near the optimal lay above the line $y = 0$. A chromosome encoding a point above this line must have gene 33 bound to allele 0, therefore this gene is highly selective and should be bound early. As evolution progresses and the alleles of the selective genes become nearly homogeneous in the population, additional genes become se-

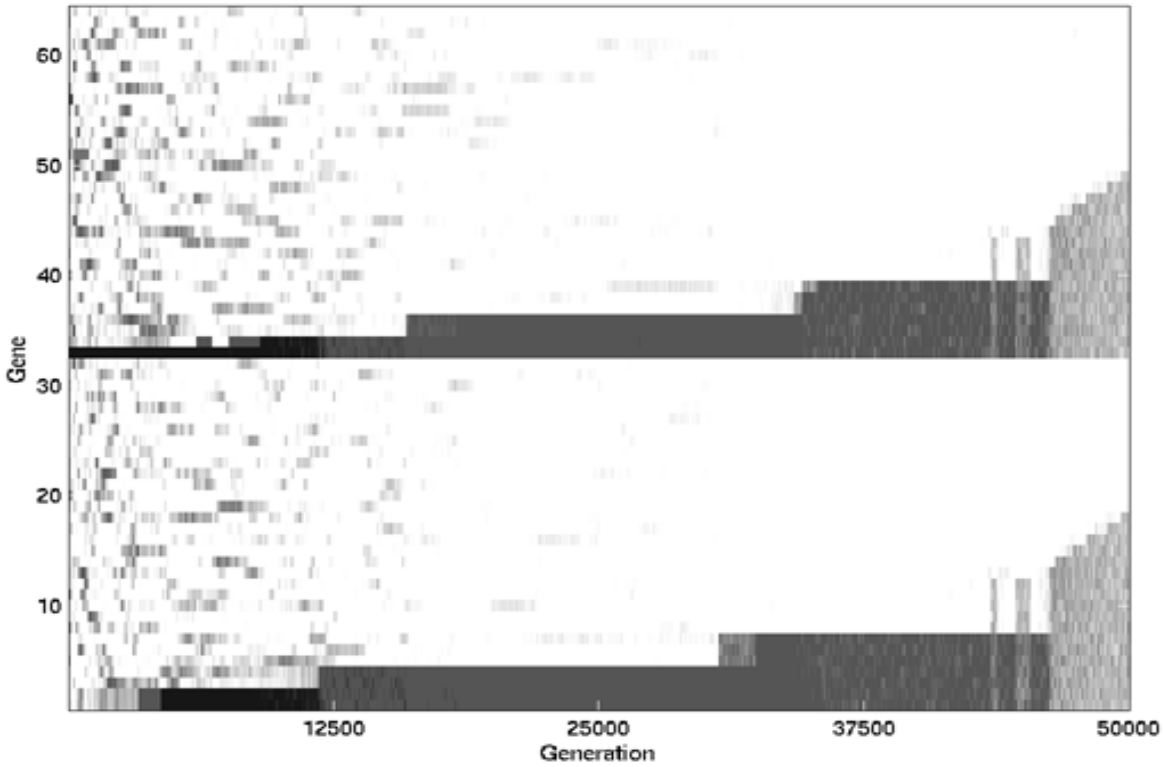


Figure 3: Histogram of Gene Order for Minimization of the Rosenbrock Function Over 50,000 Generations. Dark coloration indicates the gene was bound early in chromosome creation, light means late. TEAM identifies and solves the most selective genes first, thereby making consistent refinement possible.

lective. This process is visible in Figure 3 as a widening of the two dark bands. TEAM's focus on these new selective genes incrementally refines the point (x, y).

This way, Figure 3 illustrates how TEAM solves the problem by identifying the most important genes first. Similar processes take place in other domains where the dependencies may be less obvious, giving us important insight into the domain. To an observer, gene order can be an important clue to identifying underlying inter-gene dependencies in the problem.

4 DISCUSSION & FUTURE WORK

The overhead of maintaining the model and constructing the chromosome in TEAM is considerable compared to that of a hillclimber or a GA. However, such more intelligent evolution steps are warranted if they can generate better final solutions than other methods. In the challenging test of minimizing the Rosenbrock function, TEAM indeed produced solutions significantly better than the other algorithms.

TEAM relies on the t-test, which assumes samples are normally distributed. While TEAM is effective in the two selected problems, normality may be a risky assumption in other domains. It might therefore make sense to replace the t-test with a method that works equally well with arbitrary distributions, such as the Mann-Whitney U test (Mendenhall and Beaver, 1994). Instead of scoring chromosomes and genes based on fitness values, scores could be based on rank within chromosome/gene sets. This way it should be possible to apply TEAM reliably to a wide range of domains.

Rank-based calculations have another advantage: they limit the influence high-fitness chromosomes have during chromosome creation. Currently, if there is a small number of individuals with very high fitness, the new chromosome is constructed mostly based on their genes. However, the goal of evolution in TEAM is to produce a population that converges around a few high-fitness individuals. With rank-based calculations a larger number of chromosomes take part in construction, thereby maximizing the amount of information considered during chromosome creation.

The model maintained by TEAM has perhaps the greatest potential for improvement. The current

model is organized around pair-wise relationships between genes. This is an improvement over algorithms that deal with genes in isolation, but combinatorial problems are not limited to only pair-wise dependencies. A method for identifying and exploiting n-gene relationships is needed. Instead of recording gene pairs with an observed relationship to fitness values, we can record gene groups of arbitrary size. We can already identify groups of size two, and larger groups can be formed through expansion and combination of existing groups as additional correlations are found. The identification of gene groups could be run in parallel with population evolution. It is an interesting problem in its own right, and will be studied in detail in future work.

5 CONCLUSIONS

An advanced constructive evolutionary algorithm, TEAM, was introduced in this paper. Using statistical analysis and modeling of gene/fitness correlations, TEAM can evolve chromosomes in domains with complex gene dependencies. Experiments show that TEAM performs better than other problem-independent combinatorial optimization techniques on difficult problems. This result shows that domain information can be extracted and meaningfully applied during population evolution. Future work will focus on improved methods of information extraction and application from limited domain sampling.

Acknowledgments

Thanks to Daniel Polani and John Prior for useful discussions and code for the EuA and EuSane algorithms. This research was supported in part by the NSF under grant IIS-0083776 and by the Texas Higher Education Coordinating Board under grant ARP-003658-476-2001.

References

Bäck, T. (1992). *A User's Guide to GENEsYs 1.0.* Technical report. Department of Computer Science, University of Dortmund.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning.* Technical Report CMU-CS-94-163. Carnegie Mellon University, Pittsburgh, PA.

Davidor, Y. (1991). Epistatis Variance: A Viewpoint on GA-Hardness. *Foundations of Genetic Algorithms 1*, 23-35. Morgan Kaufmann, San Mateo, CA.

De Jong, K. A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral dissertation, University of Michigan, *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No. 76-9381).

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* University of Michigan Press, Ann Arbor, MI.

Kirkpatrick, S., and Sherrington, D. (1988). Infinite range models of spin-glasses. *Physical Review B*, 17:4384-4403.

Mendenhall, W., and Beaver, R. (1994). *Introduction to Probability and Statistics - 9th edition.* Duxbury Press, International Thomson Publishing, Pacific Grove, CA.

Polani, D., and Miikkulainen, R. (2000). Eugenic Neuro-evolution for reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO-2000, Las Vegas, NV), 1041-1046. Morgan Kaufmann, San Francisco, CA.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C.* Cambridge University Press. Second Edition.

Prior, J. W. (1998). *Eugenic Evolution for Combinatorial Optimization.* Master's thesis, Technical Report AI98-268. Department of Computer Sciences, The University of Texas at Austin.

Syswerda, G. (1993). Simulated Crossover in Genetic Algorithms. *Foundations of Genetic Algorithms 2*, 239-255. Morgan Kaufmann, San Mateo, CA.

An adaptive penalty scheme in genetic algorithms for constrained optimization problems

Helio J.C. Barbosa
LNCC/MCT
Rua Getulio Vargas 333
25651 070 Petropolis RJ, BRAZIL
hcbm@lncc.br

Afonso C.C. Lemonge
Departamento de Estruturas
Faculdade de Engenharia
Universidade Federal de Juiz de Fora
36036 330 Juiz de Fora MG, BRAZIL
lemonge@numec.ufjf.br

Abstract

A parameter-less adaptive penalty scheme for genetic algorithms applied to constrained optimization problems is proposed. Besides being simple, adaptive and not requiring any parameter, it provides a different penalty parameter for each constraint. The performance of this new scheme is examined using several test-problems from the evolutionary computation literature.

1 INTRODUCTION

The design and implementation of a robust genetic algorithm (GA) for constrained problems is not an easy task. Several possibilities have been explored by the evolutionary computation community and they can be roughly grouped as: 1) penalty techniques, 2) repair methods, 3) special decoders, 4) special operators, 5) selection techniques, 6) hybrid methods and 7) other methods.

Repair methods use domain knowledge in order to move infeasible offspring into the feasible set. However there are situations when it is very expensive, or even impossible, to construct such a repair operator, drastically reducing the range of applicability of repair methods. Like repair methods, the design of special decoders[1] that always extract a feasible phenotype from a given genotype is not trivial in general and cannot always be done. In special situations genetic operators can be constructed so that a feasible offspring is always generated from feasible parents[2]. Hybrid methods, combining mathematical programming and evolutionary techniques have also been developed[3, 4]. Modified selection schemes used alone [5] or in conjunction with additional features [6] are also proposed. Under the heading "other methods" one usually collects those which cannot be clearly classified in one of the previous groups such as the use of co-evolution (see[7, 8]). For other methods found in the evolutionary computation literature see[2, 9, 10, 4] and references therein.

Penalty techniques range from simple naive schemes (like the "death penalty": any infeasible offspring is just discarded with no consideration for its potential information content) to penalty schemes involving from one to several parameters. Those parameters can remain constant (the most common case) or be dynamically varied along the evolutionary process according to an exogenous schedule[11, 12] or an adaptive procedure[13]. Penalty methods, though quite general, require considerable domain knowledge and experimentation in each particular application in order to be effective.

In contrast with previous approaches where a single penalty parameter is used for all constraints in a given problem, an adaptive scheme is proposed here which automatically sizes the penalty parameter corresponding to *each* constraint along the evolutionary process.

In the next section the main penalty techniques found in the literature are summarized. Section 3 presents the adaptive scheme proposed and Section 4 discusses several numerical experiments with test-problems. The paper ends with conclusions in Section 5.

2 THE CONSTRAINED OPTIMIZATION PROBLEM

A standard constrained optimization problem in R^n can be thought of as the minimization of a given objective function $f(x)$, where $x \in R^n$ is the vector of design/decision variables, subject to inequality constraints $g_p(x) \geq 0$, $p = 1, 2, \ldots, \bar{p}$ as well as equality constraints $h_q(x) = 0$, $q = 1, 2, \ldots, \bar{q}$. Additionally, the variables may be subject to bounds $x_i^L \leq x_i \leq x_i^U$ but this type of constraint is trivially enforced in a GA and need not be considered here.

2.1 Penalty functions

Although a few papers [14] do consider a *multiplicative* penalty function, in the widely used additive penalty formulation one is led to the unconstrained minimization of a modified objective function

$$F(x) = f(x) + penal(x) \tag{1}$$

where the penalty function $penal(x)$ is zero if x is a feasible solution and greater than zero otherwise. It is useful to define the amount of violation of the j-th constraint by the solution $x \in R^n$ as

$$v_j(x) = \begin{cases} |h_j(x)| & \text{equality case} \\ \max\{0, -g_j(x)\} & \text{inequality case} \end{cases} \tag{2}$$

It is also common to design penalty functions that grow with the vector of violations $v(x) \in R^m$ where $m = \bar{p} + \bar{q}$ is the number of constraints to be penalized. The most popular penalty function is given by

$$penal(x) = k \sum_{j=1}^{m} (v_j(x))^2 \tag{3}$$

where k is the penalty parameter. It can be shown that as $k \to \infty$, the corresponding solution x_k^* of the penalized problem tends to the solution x^* of the original problem. Although it is easy to obtain the unconstrained problem, the definition of a good penalty parameter k is usually a very time-consuming trial-and-error process.

Among the many penalty techniques found in the literature[15, 3, 16] some of them – more closely related to the work presented here – will be briefly discussed below.

2.1.1 Some methods in the literature

Homaifar *et al.* [17] write the fitness function as in (1) – (3) but allow the user to define several levels of constraint violation in such a way that the penalty coefficients grow as higher levels of violation are reached. As a result, the method requires a penalty parameter k_j^l for the l-th level of violation of the j-th constraint. This is an attractive strategy because, at least in principle, it allows for a good control of the penalization process. The weakness of this method is the large number of parameters that must be set by the user for each problem.

Joines & Houck [11] introduce dynamic penalty parameters and the fitness function $F(x)$ can be written as in (1) – (3) with the penalty parameter, given by $k = (C \times t)^\alpha$, increasing with the generation number t.

Bean & Alouane [18] use (1) – (3) but with the penalty parameter $k = \lambda(t)$ adapted at each generation by the following rules:

$$\lambda(t+1) = \begin{cases} (\frac{1}{\beta_1})\lambda(t) & \text{if } b^i \in \mathcal{F} \text{ for all } t-g+1 \le i \le t \\ \beta_2\lambda(t) & \text{if } b^i \notin \mathcal{F} \text{ for all } t-g+1 \le i \le t \\ \lambda(t) & \text{otherwise} \end{cases}$$

where b^i is the best element at generation t, $\mathcal{F}$ is the feasible region, $\beta_1 \neq \beta_2$ and $\beta_1, \beta_2 > 1$.

Coit *et al.* [13], use the fitness function:

$$F(x) = f(x) + (F_{feas}(t) - F_{all}(t)) \sum_{j=1}^{m} (v_j(x)/v_j(t))^\alpha$$

where $F_{all}(t)$ corresponds to the best solution until the generation t (without penalty), F_{feas} corresponds to the best feasible solution and α is a constant.

Schoenauer & Xhantakis [19] handle constrained problems in stages: (i) first evolve a randomly generated population considering only the first constraint until a certain percentage of the population is feasible with respect to that constraint; (ii) the final population of the first stage of the process is used in order to optimize with respect to the second constraint. During this stage, the elements that had violated the previous constraint are removed from the population, (iii) the process is repeated until all the constraints are processed. This strategy becomes less attractive as the number of constraints grows and is potentially dependent on the order in which the constraints are processed.

Le Riche *et al.* [20] create two sets of candidate solutions where one of them is evaluated with a penalty parameter k_1 and the other with a parameter k_2. With $k_1 \gg k_2$ there are two different levels of penalization and there is a higher chance of maintaining feasible as well as infeasible individuals in the population and to get offsprings near the boundary between the feasible and infeasible regions.

Powell & Skolnick [21] propose a method of superiority of feasible points where each candidate solution is evaluated by the following expression:

$$F(x) = f(x) + r \sum_{j=1}^{m} v_j(x) + \theta(t, x)$$

where r is a constant. The main assumption is that any feasible solution is better than any infeasible solution. This assumption is enforced by a convenient definition of the function $\theta(t, x)$. Deb [6] modified this scheme by introducing tournament selection coupled with the fitness function:

$$F(x) = \begin{cases} f(x) & \text{if } x \text{ is feasible} \\ f_{max} + \sum_{j=1}^{m} v_j(x) & \text{otherwise} \end{cases} \tag{4}$$

where f_{max} – the function value of the worst feasible solution in the population – replaces $f(x)$, leading to a higher

(potentially excessive) penalty. However, Deb's[6] constraint handling scheme (which also involves niching and a controlled mutation) works very well for his real-coded GA, but not for the binary-coded one.

Hamida & Schoenauer [22] propose an adaptive algorithm accounting for the proportion of feasible individuals in the population, using a seduction/selection strategy to mate feasible and infeasible individuals and a selection scheme to favor a fraction of all feasible individuals.

Runarsson & Yao [5] propose a novel approach to balance objective and penalty function values by a stochastic ranking. Using a real coding, very good results are obtained for continuous optimization problems.

Recently, Wright & Farmani [23] proposed a method that requires no parameters and represents the constraint violation by a single infeasibility measure.

3 THE PROPOSED METHOD

In this work a method without any type of user defined penalty parameter is proposed. An adaptive scheme is developed which uses information from the population such as the average of the objective function and the level of violation of each constraint during the evolution.

The fitness function proposed is written as:

$$F(x) = \begin{cases} f(x), & \text{if } x \text{ is feasible,} \\ \overline{f}(x) + \sum_{j=1}^{m} k_j v_j(x) & \text{otherwise} \end{cases}$$

where

$$\overline{f}(x) = \begin{cases} f(x), & \text{if } f(x) > \langle f(x) \rangle, \\ \langle f(x) \rangle & \text{otherwise} \end{cases} \tag{5}$$

and $\langle f(x) \rangle$ is the average of the objective function values in the current population. In the Figure 1 feasible as well as infeasible solutions are shown. Among the 6 infeasible solutions, the individuals #3, #4, #5 and #6 have their objective function values (represented by opened circles), less than the average objective function and, according to the proposed method, have $\overline{f}$ given by $\langle f(x) \rangle$. The solutions #1 and #2 have objective function values which are worst than the population average and thus have $\overline{f}(x) = f(x)$.

The penalty parameter is defined at each generation by:

$$k_j = |\langle f(x) \rangle| \frac{\langle v_j(x) \rangle}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} \tag{6}$$

and $\langle v_l(x) \rangle$ is the violation of the l-th constraint averaged over the current population. Denoting by *pop* the population size, one could also write

$$k_j = \frac{|\sum_{i=1}^{pop} f(x^i)|}{\sum_{l=1}^{m} [\sum_{i=1}^{pop} v_l(x^i)]^2} \sum_{i=1}^{pop} v_j(x^i) \tag{7}$$

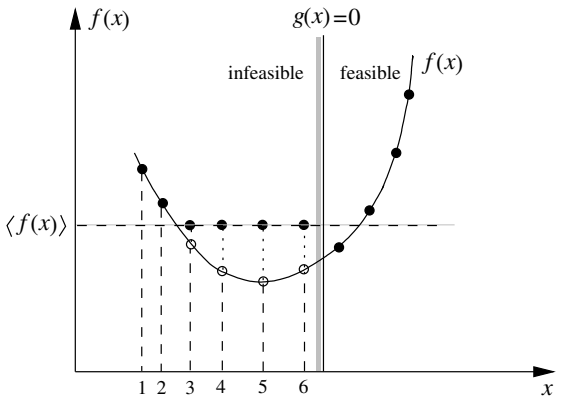


Figure 1: The definition of the function $\overline{f}$.

With the proposed definition one can prove the following:

Property – An individual whose j-th violation equals the average of the j-th violation in the current population for all j, has its fitness value given by:

$$F(\tilde{x}) = \begin{cases} f(\tilde{x}) + |\langle f(x) \rangle| & \text{if } f(\tilde{x}) > \langle f(x) \rangle \\ \langle f(x) \rangle + |\langle f(x) \rangle| & \text{otherwise} \end{cases}$$

Proof – In fact, let $\tilde{x}$ be such an element. By definition,

$$F(\tilde{x}) = \overline{f}(\tilde{x}) + \sum_{j=1}^{m} \frac{|\langle f(x) \rangle| \langle v_j(x) \rangle}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} v_j(\tilde{x}).$$

But, by hypothesis, $v_j(\tilde{x}) = \langle v_j(x) \rangle$ for all j leading to

$$\begin{aligned} F(\tilde{x}) &= \overline{f}(\tilde{x}) + \frac{|\langle f(x) \rangle|}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} \sum_{j=1}^{m} [\langle v_j(x) \rangle]^2 \\ &= \overline{f}(\tilde{x}) + |\langle f(x) \rangle| \end{aligned}$$

and the results follow from eq. (5).

In the next section several examples from the literature are considered in order to test the robustness of the proposed adaptive parameter-less scheme. It should be emphasized that the accuracy of the final results of the search depends also on other components of the algorithm not considered here – such as coding, operators and selection scheme – besides the penalization procedure itself. Good results reported in the literature (such as in [5]) are obtained with real coding, more sophisticated selection schemes and more effective genetic operators. The need for better operators in order to obtain good results in constrained continuous optimization has already been pointed out by Hamida & Petrowski [24].

4 NUMERICAL EXPERIMENTS

In order to investigate the robustness of the proposed penalty procedure, several optimization problems from the literature are solved using a simple generational GA with Gray code, rank-based selection and elitism. The operation of recombination was applied with probability $p_r = 0.8$. Standard one-point, two-point and uniform crossover operators were applied each one with its respective relative probability (in this work, $p_c^1 = 0.2, p_c^2 = 0.4$ and $p_c^u = 0.4$). Mutation was applied bit-wise to the offsprings with rate $p_m = 0.03$. The equality constraints were converted into one inequality constraint bounding the absolute value of the degree of violation by 0.0001 (That is, $|h(x)| \leq 0.0001$).

4.1 Test-problem 1

In the first example to be investigated, from [6], the function to be minimized and the constraints are given respectively by

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2,$$
$$4.84 - (x_1 - 0.05)^2 - (x_2 - 2.5)^2 \geq 0,$$
$$x_1^2 + (x_2 - 2.5)^2 - 4.84 \geq 0$$

The search space is bounded by $0 \leq x_i \leq 6, i = 1, 2$ and the optimum solution is $(x_1, x_2) = (2.246826, 2.381865)$. with a function value $f(x) = 13.59085$. The number of generations allowed was 50 and the best solution found was $(2.2468493, 2.3823017)$ corresponding to $f(x) = 13.59085$, the worst one was $(1.5048537, 4.1170674)$ with $f(x) = 152.54840$ and the average value was $f(x) = 30.7488$. In this Test-problem 50 independent runs were performed with a population size equal to 50 and each variable coded with 20 bits. Table 1 presents a comparison of results for this test-problem.

	This study	Ref.[6]
best	13.59085	13.59658
worst	152.54840	244.11616

Table 1: Comparison of results on Test-problem1. The optimum value is 13.59085.

One can note that the results found for the Test-problem 1 are better than those found in the reference [6] using the same number of function evaluations.

4.2 Test-problem 2

In this test problem, from Michalewicz[25], the function

$$f(x, \mathbf{y}) = 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5$$

is to be minimized over the set $y_3 \leq 1$, $y_4 \leq 1$, $y_5 \leq 2$, $x \geq 0$ and $y_i \geq 0$ for $1 \leq i \leq 5$. subject to

$$x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 \leq 16$$
$$-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 \leq -1$$
$$2x + 0.5y_1 + 0.2y_2 - 3y_3 - y_4 - 4y_5 \leq 24$$
$$0.2x + 2y_1 + 0.1y_2 - 4y_3 + 2y_4 + 2y_5 \leq 12$$
$$-0.1x - 0.5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 \leq 3$$

The global solution is $(x, \mathbf{y}^*) = (0, 6, 0, 1, 1, 0)$ with $f(x, \mathbf{y}^*) = -11$. Using a population size of 70, three sets of 10 independent runs allowing for 1000, 5000 and 20000 generations were performed. Each variable was coded with 25 bits and the results are shown in the Table 2. Michalewicz [25], using GENECOP, was able to find the optimum solution in 1000 generations for all runs. However, it must be noted that GENECOP is designed to maintain feasibility for that particular constraint set (*linear* inequalities).

	maxgen		
	1000	5000	20000
x	0.000001907	0.000000715	0.000000000
y_1	6.000138938	5.998155534	5.999983966
y_2	0.002714396	0.000119925	0.000008345
y_3	0.999999881	1.000000000	1.000000000
y_4	0.989082306	0.999896646	0.999980778
y_5	0.002151549	0.000607729	0.000004590
best	-10.98587	-10.99879	-10.99997
average	-10.96019	-10.99447	-10.99965
worst	-10.90891	-10.98327	-10.99887

Table 2: Comparison of results in the Test-problem 2.

4.3 Test-problem 3

The previous test-problems presented only continuous variables. The present one, from [26], involves continuous as well as discrete variables. The function to be maximized is

$$f(x_1, x_2, x_3, y_1, y_2) = -5.357854x_1^2 -$$
$$0.835689y_1x_3 - 37.29329y_1 + 40792.141$$

subject to

$$a_1 + a_2y_2x_3 + a_3y_1x_2 - a_4x_1x_3 \leq 92$$
$$a_5 + a_6y_2x_3 + a_7y_1y_2 - a_8x_1^2 - 90 \leq 20$$
$$a_9 + a_{10}x_1x_3 + a_{11}y_1x_1 - a_{12}x_1x_2 - 20 \leq 5$$

The range of the continuous variables x_1, x_2 and x_3 is $[27, 45]$ and the ranges of the integer variables y_1 and y_2 are $[78, 102]$ and $[33, 45]$, respectively. The coefficients a_1 to a_{12} are given in the Table 3.

$a_1 = 85.334407$	$a_5 = 80.51249$	$a_9 = 9.3009610$
$a_2 = 0.0056858$	$a_6 = 0.0071317$	$a_{10} = 0.0047026$
$a_3 = 0.0006262$	$a_7 = 0.0029955$	$a_{11} = 0.0012547$
$a_4 = 0.0022053$	$a_8 = 0.0021813$	$a_{12} = 0.0019085$

Table 3: Coefficients for Test-problem 3.

The global solution is, for any value of x_2 and y_2, $(27, x_2, 27, 78, y_2)$. The number of bits per variable was 25, except for the variables y_1 and y_2 which were coded with 8 bits each. The number of generations allowed was 100 and the population size was set to 250. All 10 runs produced the optimal solution with $f = 32217.4$. In Table 4 a comparison is made between some of the results obtained by Costa & Oliveira [26] for this problem. The first column corresponds to a standard penalty scheme – eq. (1) – (3) – with $k = 10^3$, the second one to Deb's penalty scheme (eq. 4) and the last one to the present study. The search process was terminated by Costa & Oliveira [26] when: (i) 1000 generations were performed or (ii) no improvement was observed after 50 generations or (iii) all the population converged to a single solution. The Table 4 shows the average number of function evaluations (avgnf) and the percentage of runs finding the optimal solution (% success).

	standard ($k = 10^3$)	Deb's	This study
avgnf	37167	35255	25000
% success	100	100	100

Table 4: Comparison of results on Test-problem 3.

4.4 Test-problem 4

This test corresponds to a mechanical design minimization problem studied by Deb [6]. The design variables are $\{h, l, t, b\}$, with bounds $0.125 \leq h \leq 10$ and $0.1 \leq l, t, b \leq 10$, and the objective function is

$$f_w = 1.10471h^2l + 0.04811tb(14.0 + l)$$

subject to

$$13,600 - \tau(x) \geq 0, \quad 30,000 - \sigma(x) \geq 0,$$
$$b - h \geq 0,\ P_c(x) - 6,000 \geq 0,\ 0.25 - \delta(x) \geq 0$$

The expressions for $\tau(x)$, $\sigma(x)$, $P_c(x)$ and $\delta(x)$ are:

$$\tau(x) = \sqrt{\left(\tau^{'}(x)\right)^2 + \left(\tau^{''}(x)\right)^2 + l\tau^{'}(x)\tau^{''}(x)/c}$$
$$c = \sqrt{0.25\left(l^2 + (h+t)^2\right)}, \quad \sigma(x) = \frac{504000}{t^2b}$$
$$P_c(x) = 64746.022\left(1 - 0.0282346t\right)tb^3$$
$$\delta(x) = \frac{2.1952}{t^3b}, \quad \tau^{'}(x) = \frac{6000}{\sqrt{2}hl}$$
$$\tau^{''}(x) = \frac{6000\left(14 + 0.5l\right)c}{2\left\{0.707hl\left(l^2/12 + 0.25\left(h+t\right)^2\right)\right\}}$$

In [6] Deb investigates two situations in order to compare the results for this example. The first one corresponds to a binary coded GA with standard penalty scheme – eq. (1) - (3) – and four constant penalty coefficients ($k = 10^0$, $k = 10^1$, $k = 10^3$ and $k = 10^6$). The population size was set to 80, the maximum number of generations was 500 and 50 independent runs were performed. Each variable was coded with 10 bits and the total length of the chromosome was 40. The Table 5 shows the comparison of results and it is clear that in this study better values were found.

	best	worst
$k = 10^0$	2.41324	483.50177
$k = 10^1$	3.14206	7.45453
$k = 10^3$	3.38277	10.65891
$k = 10^6$	3.72929	9.42353
This study	2.39623	3.39956

Table 5: Comparison of results on Test-problem 4 using standard penalty and the proposed technique.

The second situation (referred to as the "welded beam revisited" by Deb) corresponds to a real coded GA proposed by Deb where, among other features, niching was used. The population size was set to 80. Two series of 50 runs were performed allowing for 500 and 4000 generations respectively. Each variable was coded with 30 bits generating a chromosome 120 bits long. Results for this test are shown in Table 6.

	maxgen	This study	Ref. [6]	
			w/o niching	with niching
best	500	2.38352	2.44271	2.38119
	4000	2.38159	—	2.38119
worst	500	3.73060	7.44425	2.64583
	4000	2.95533	—	2.38355

Table 6: Comparison of results on Test-problem 4 using Deb's real coded GA and the proposed technique.

It is clear that Deb's results only improve when niching is added to his penalty scheme.

4.5 The G-Suite

In this section the 11 well known G1-G11 test-problems presented by Koziel & Michalewicz [1] are considered. The G-Suite is made up of distinct kinds of functions and

involves constraints given by linear inequalities LI, nonlinear equalities NE and nonlinear inequalities NI. The summary of the Test-cases is presented in the Table 7 where the column "n" shows the number of parameters, "a" indicates the number of active constraints at the optimum solution and f^* denotes the known optimum value of f. More details of each of these problems can be found in [1] and [5]. An extended discussion involving each one of these problems and different techniques from the evolutionary computation literature can be found in [27].

Three set of experiments were performed using a population size of 70. For the first experiment 20 independent runs were performed with the maximum number of generations set to 5000. In the second experiment the maximum number of generations is set to 20000. Finally, the third experiment consists of 10 independent runs (with 5000 generations) where the best solution obtained in the first experiment is introduced in the initial population.

Name	n	Type of $f(x)$	LI	NE	NI	a	f^*
G1	13	quadratic	9	0	0	6	-15.0
G2	20	nonlinear	9	0	6	1	0.803553
G3	10	polynomial	0	1	0	1	1.0
G4	5	quadratic	0	0	6	2	-30655.5
G5	4	cubic	2	3	0	3	5126.4981
G6	2	cubic	0	0	2	2	-6961.8
G7	10	quadratic	3	0	5	6	24.306
G8	2	nonlinear	0	0	2	0	0.0958250
G9	7	polynomial	0	0	2	0	680.63
G10	8	linear	3	0	3	6	7049.33
G11	2	quadratic	0	1	0	1	0.75

Table 7: Summary of 11 Test Cases.

A comparison of the results obtained with the simple adaptive procedure proposed here with those obtained by Koziel & Michalewicz [1] is made in the Tables 8, 9, 10, 11, 12 and 13 where the best results are indicated in **boldface**.

It can be noted that the proposed scheme provides better results than those obtained using the more complex homomorphous mapping technique of Koziel & Michalewicz [1].

The same observation applies to the results recently obtained by Wright & Farmani [23] using a binary code.

5 CONCLUSIONS

A new simple adaptive parameter-less penalty scheme for the solution of constrained optimization problems via genetic algorithms has been proposed. Its main feature, besides being adaptive and not requiring any parameter, is to automatically define a different penalty parameter for each constraint.

In all problems tested so far (including tests not shown here) the procedure has produced good results when compared to the literature available. It must also be emphasized that such good results were obtained in spite of the use of an otherwise very naive binary coded genetic algorithm with standard operators. Good results reported in the literature are often obtained with real coding, more sophisticated selection schemes, more aggressive genetic operators and by exploring the particular structure of the constraint set.

The procedure proposed here is simpler, can be implemented for binary coded genetic algorithms leading to good results for mixed continuous-discrete optimization problems. Additional tests in larger real-world applications are being conducted with good results so far.

Acknowledgements

The authors acknowledge the support received from CNPq (301233/86-1 and 475398/2001-7) and FAPEMIG (TEC-692/99). The authors would also like to thank the reviewers for the corrections and suggestions which helped improve the quality of the paper.

References

[1] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[2] M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In H-M. Voigt; W. Ebeling; I. Rechenberg and H-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141, pages 245–254, Berlin, 1996. Springer-Verlag. LNCS.

[3] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[4] J.-H. Kim and H. Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 2(1):129–140, 1997.

[5] T.P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[6] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, June 2000.

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

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

[9] R. Hinterding and Z. Michalewicz. Your brains and my beauty: Parent matching for constrained optimization. In *Proc. of the Fifty Int. Conf. on Evolutionary Computation*, pages 810–815, Alaska, May 4-9 1998.

[10] S. Koziel and Z. Michalewicz. A decoder-based evolutionary algorithm for constrained optimization problems. In T. Bäck; A.E. Eiben; M. Schoenauer and H.-P. Schwefel, editors, *Proc. of the Fifth Parallel Problem Solving from Nature*, Amsterdam, September 27-30 1998. Springer-Verlag. Lecture Notes in Computer Science.

[11] J.A Joines and C.R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In Z. Michalewicz; J.D. Schaffer; H.-P. Schwefel; D.B. Fogel and H. Kitano, editors, *Proc. of the First IEEE Int. Conf. on Evolutionary Computation*, pages 579–584, June 19-23 1994.

[12] Z. Michalewicz and N. Attia. Evolutionary optimization of constrained problems. In *Proc. of the Third Annual Conference on Evolutionary Programming*, pages 98–108, River Edge, 1994. World Scientific Publishing.

[13] D.W. Coit, A.E. Smith, and D.M. Tate. Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing*, 6(2):173–182, 1996.

[14] S.E. Carlson and R. Shonkwiler. Annealing a genetic algorithm over constraints. In *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, pages 3931–3936, 1998.

[15] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation. In *Proc. of the 4th Int. Conf. on Evolutionary Programming*, pages 135–155, Cambridge, MA, 1995. MIT Press.

[16] Z. Michalewicz, D. Dasgupta, R.G. Le Riche, and M. Schoenauer. Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal*, 30(2):851–870, 1996.

[17] H. Homaifar, S.H.-Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.

[18] J.C. Bean and A.B. Alouane. A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Departament of Industrial and Operations Engineering, The University of Michigan, 1992.

[19] M. Schoenauer and S. Xanthakis. Constrained GA optimization. In S. Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 573–580, Los Altos, CA, 1993. Morgan Kaufmann Publishers.

[20] R.G. Le Riche, C. Knopf-Lenoir, and R.T. Haftka. A segregated genetic algorithm for constrained structural optimization. In L.J. Eshelman, editor, *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pages 558–565, Pittsburgh, PA., July 1995.

[21] D. Powell and M.M. Skolnick. Using genetic algorithms in engineering design optimization with nonlinear constraints. In S. Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 424–430, San Mateo, CA, 1993. Morgan Kaufmann.

[22] S.B. Hamida and M. Schoenauer. An adaptive algorithm for constrained optimization problems. In *Parallel Problem Solving from Nature – PPSN VI*, volume 1917, pages 529–538, Berlim, 2000. Springer-Verlag. Lectures Notes in Computer Science.

[23] J.A. Wright and R. Farmani. Genetic algorithms: A fitness formulation for constrained minimization. In *Proc. of the Genetic and Evolutionary Computation Conference – GECCO 2001*, pages 725–732, San Francisco, CA., 2001. Morgan Kaufmann.

[24] S. B. Hamida and A. Petrowski. The need for improving the exploration operators for constrained optimization problems. In *2000 Congress on Evolutionary Computation*, pages 1176–1183, San Diego, CA, USA, July 2000. IEEE Service Center.

[25] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

[26] L. Costa and P. Oliveira. Evolutionary algorithm approach to the solution of mixed integer non-linear programming problems. *Computers and Chemical Engineering*, 25:257–266, 2001.

[27] Z. Michalewicz and D.B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, 1999.

Function	Experiment 1		
	worst	best	average
G1	**−15.00**	**−15.00**	**−15.00**
G2	0.6002978	0.7724640	0.7031971
G3	0.9391756	0.9939078	0.9757277
G4	**−30660.76**	**−30665.24**	**−30663.40**
G5	**6040.595**	**5126.571**	**5389.364**
G6	**−6961.779**	**−6961.796**	**−6961.789**
G7	42.01618	**24.86371**	29.86465
G8	**0.0725015**	**0.0958250**	**0.0926157**
G9	**682.1562**	**680.7590**	**681.4076**
G10	**10002.93**	**7086.404**	**8161.997**
G11	0.7579745	**0.75**	0.7503349

Table 8: Experiment 1.

Function	Experiment 1		
	worst	best	average
G1	−14.0566	−14.7207	−14.4609
G2	**0.78427**	**0.79506**	**0.79176**
G3	**0.9917**	**0.9983**	**0.9965**
G4	−30617.0	−30662.5	−30643.8
G5	—	—	—
G6	−4236.7	−6901.5	−6191.2
G7	**38.682**	25.132	**26.619**
G8	0.0291434	**0.095825**	0.0871551
G9	682.88	681.43	682.18
G10	11894.5	7215.8	9141.7
G11	**0.75**	**0.75**	**0.75**

Table 9: Experiment 1 (Koziel & Michalewicz [1]).

Function	Experiment 2		
	worst	best	average
G1	**−15.00**	**−15.00**	**−15.00**
G2	0.6499022	0.7918570	0.7514353
G3	**0.9983935**	1.000307	**0.9997680**
G4	**−30664.91**	**−30665.51**	**−30665.29**
G5	**6040.595**	**5126.571**	**5389.347**
G6	**−6961.796**	**−6961.796**	**−6961.796**
G7	33.07581	24.85224	27.90973
G8	**0.0795763**	**0.0958250**	**0.0942582**
G9	**681.6396**	**680.6678**	**680.9640**
G10	**9977.767**	**7080.107**	**8018.938**
G11	**0.75**	**0.75**	**0.75**

Table 10: Experiment 2.

Function	Experiment 2		
	worst	best	average
G1	−14.6154	−14.7864	−14.7082
G2	**0.79119**	**0.79953**	**0.79671**
G3	0.9978	**0.9997**	0.9989
G4	−30643.8	−30645.9	−30655.3
G5	—	—	—
G6	−5473.9	−6952.1	−6342.6
G7	**25.069**	**24.62**	**24.826**
G8	0.0291438	**0.095825**	0.0891568
G9	683.18	680.91	681.16
G10	9659.3	7147.9	8163.6
G11	**0.75**	**0.75**	**0.75**

Table 11: Experiment 2 (Koziel & Michalewicz [1]).

Function	Experiment 3		
	worst	best	average
G1	**−15.00**	**−15.00**	**−15.00**
G2	0.7729277	0.7780233	0.7741776
G3	**0.9973952**	**0.9997135**	**0.9987124**
G4	**−30665.25**	**−30665.51**	**−30665.39**
G5	**5126.571**	**5126.571**	**5126.571**
G6	**−6961.796**	**−6961.796**	**−6961.796**
G7	**24.86371**	**24.86130**	**24.86346**
G8	0.0918033	**0.0958250**	0.0940601
G9	**680.7590**	**680.7222**	**680.7461**
G10	**7082.667**	**7080.328**	**7081.146**
G11	**0.75**	**0.75**	**0.75**

Table 12: Experiment 3.

Function	Experiment 3		
	worst	best	average
G1	−14.5732	−14.7184	−14.6478
G2	**0.78279**	**0.79486**	**0.78722**
G3	0.996	0.9978	0.997
G4	−30645.6	−30661.5	−30653.1
G5	—	—	—
G6	−6390.6	−6944.4	−6720.4
G7	26.182	25.09	25.545
G8	**0.0958246**	**0.0958250**	**0.0958248**
G9	683.58	681.72	682.56
G10	7685.8	7321.2	7498.6
G11	**0.75**	**0.75**	**0.75**

Table 13: Experiment 3 (Koziel & Michalewicz [1]).

Expediting Genetic Search with Dynamic Memory

Jason S. Byassee and Keith E. Mathias
TRW Systems
16201 E. Centretech Pkwy.
Aurora, CO 80011
jason.byassee, keith.mathias@auc.trw.com

Abstract

Exploitation of domain knowledge can expedite the process of finding solutions to new problems. This research is focused on a distributed memory system which maintains a dynamic knowledge base, in the form of memory cells, that are employed to improve search performance over time. Short-term and long-term memory models are analyzed in the context of the distributed memory system. Results indicate that genetic search performance is significantly impacted by the quantity and quality of information that is maintained in memory.

1 Introduction

Many real-world optimization problems are time sensitive, where unbounded time to find the optimal solution is not practical. In these instances, execution time can be expedited by leveraging domain knowledge, providing a "starting point" for the search algorithm. Extensive research has been performed in the context of incorporating *a-priori* knowledge to improve genetic search performance [4, 5]. However, it is still unclear how the quantity and quality of information that constitutes the knowledge base affect search performance. This provides the motivation for examining the impact of short-term and long-term memory on genetic search performance, in terms of both quantitative and qualitative metrics.

We have developed a distributed memory system (DMS) that serves as the test environment for our analysis. In this DMS, genetic search is employed as the search mechanism on the system nodes. The nodes operate independently and simultaneously, sampling and solving different pattern matching problems from a shared library. The DMS incorporates a dynamic pool of memory cells that is shared between nodes. However, information from any one cell is available at only one node at a time. The memory cells evolve with continuous feedback from each independent genetic search. Each node in the DMS employs genetic search to solve an independent problem as opposed to all nodes working together to solve a single problem, thereby differentiating this work from most parallel genetic algorithm research.

A secondary goal for each search is to improve the aggregate search performance of all nodes by sharing information about problems that may be encountered by other nodes over time. Our objective is to compare genetic search performance when knowledge is shared via short- or long-term memory (of limited size) for initial seeding rather than random initialization.

2 Distributed Memory System

Simple bit-pattern recognition problems form the bounded problem library for this work.[1] The system is trained to recognize a set of bit string patterns in a pattern library by evolving and maintaining dynamic memory cells that may be shared between nodes via communication. The memory cells (in the form of bit string patterns) are evolved by local genetic search and fed back into the system.

2.1 Operation

This DMS consists of two core operations. The first is genetic search, taking place simultaneously and continuously on each node. The second involves mobile

[1] The GA may not be the best choice for simple pattern matching problems. This application was selected so as to focus on performance metrics in the context of exploiting memory for subsequent search. We anticipate using this system for complex pattern recognition tasks in the future.

agents that circulate the memory cells between independent nodes. While the memory cells can travel to any node in the system, a cell is only useful when resident on a given node. The nodes perform pattern matching continuously. Each node has access to a pattern library, much like the immune library in Forrest, et. al., [3]. Random samples from the pattern library are taken at each node. The resident memory cells in the input queue on each respective node are compared against a selected bit string. If a match is found, a new sample is taken and the process is repeated. If no match is found, the resident memory cells are used as the initial population for genetic search.

In this system, autonomous mobile agents circulate the memory cells. From the perspective of each node, agents continuously arrive, deposit memory cells, retrieve new memory cells and transport them to new nodes (Figure 1). Meanwhile, bit strings are continuously sampled from the pattern library. At the instant that a sample is taken, the first 50 memory cells that reside in the local input queue are removed, comprising the initial population for genetic search. If the initial population is less than 50, the memory cells that reside in the local output queue are removed and added to the initial population. Finally, random bit strings are generated to satisfy any remaining discrepancy. With a complete initial population, genetic search begins for the string to match the sampled pattern.

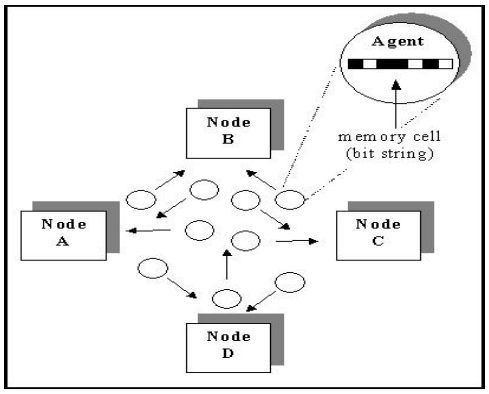


Figure 1: Agent transportation of memory cells.

2.2 Search Algorithm

The CHC adaptive search algorithm [1] is a generational genetic algorithm that has been shown to yield very good performance in optimizing a wide variety of test problems with little or no parameter tuning [6, 7]. Therefore, CHC is used as the genetic search component of our DMS. CHC begins the search process by generating $2 * N$ random samples and selecting the best N samples, where N is the population size.[2] Each of the individuals in the population is then randomly paired to form potential mating pairs. The Hamming distance between the potential mates is measured and compared with an incest threshold. If the Hamming distance is greater than the incest threshold the individuals are allowed to mate. When the mating process is completed for the $N/2$ pairs, the offspring produced compete for survival with the parent population. The best N individuals survive.

Mating in CHC is typically performed using the HUX crossover operator when a binary representation is employed. HUX is a highly disruptive crossover operator which guarantees that the offspring produced will be maximally distant (in Hamming space) from the parents. Crossover is performed by exchanging exactly half of the bits that differ between the two parents.

The incest threshold value is initially set to the expected difference between samples in the population (i.e., $L/2$, where L is the string length). The incest threshold is adaptively adjusted as search progresses. Each generation that either: a) no offspring survive or b) no matings are allowed, the incest threshold is reduced. When the incest threshold goes below 0, cataclysmic mutation is used to diverge the population. Divergence is accomplished by making $2 * N$ copies of the best individual in the population. Then 35% of the bits in all but one individual are complemented and search is restarted.

2.3 Parameters

In this DMS there are several operational parameters that can be tuned to: 1) optimize the learning curve (i.e., minimize the time to recognize the sampled patterns) and 2) minimize the memory footprint (i.e., the number of memory cells in the system). There are four parameters (*feedback percentage*, *feedback decrement*, *direct decrement* and *survival threshold*) in this DMS that directly impact memory cell survival. There are two additional parameters (*number of agents* and *agent wait time*) that effect the circulation and overall distribution of memory cells in the system.

To bound the size of the memory cell population in this system we have introduced the notion of lifetime. Memory cell lifetimes are associated with time-to-live (TTL) values. All memory cells receive the same initial TTL value when they are fed into the system. To survive, a memory cell must maintain a TTL that is greater than the survival threshold. Each time a cell

[2]The initial population in this DMS is seeded rather than randomly generated, as explained in Section 2.1.

is moved from the input to the output queue on a given node, its TTL is decremented. Figure 2 provides an overview of the operations and parameters at each node in the system. The six operational parameter descriptions [and ranges] are as follows:

1. **Number of Agents** [1 - (100*numOfNodes)] - the number of agents that exist in the system.
2. **Agent Wait Time** [100 ms - 2 sec] - the length of time that an agent waits at a node for a memory cell to become available before moving to a new node "empty handed".
3. **Feedback Percentage** [2% - 100%] - the percentage of patterns fed to the survival test from the final genetic population. String patterns are taken in order of best score to worst.
4. **Feedback Decrement** [0 - 100] - decrement applied to a memory cell's TTL when fed to the survival test from the final genetic population.
5. **Direct Decrement** [0 - 100] - decrement applied to a memory cell's TTL when fed directly from the input queue to the output queue.
6. **Survival Threshold** [1 - 100] - minimum TTL value for a memory cell's continued survival.

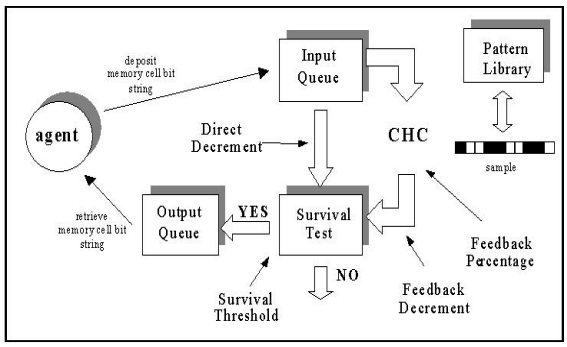


Figure 2: DMS operations and parameters.

In this DMS, the TTL parameter governs the persistence of memory cells. Short-term memory (STM) is modeled by unconditionally decrementing the TTL of memory cells each time they are used. Each cell is treated the same, regardless of its value to the system. Long-term memory (LTM) is modeled by conditional handling of the TTL value, rewarding "useful" memory cells. Upon arrival at a host, each memory cell is scored against the current pattern sample. The TTL of the memory cell is reset if its score is greater than its current TTL. Thus, in the LTM model, survival depends on possessing a high affinity (close in Hamming space) for one or more members of the pattern library.

In evaluating the STM and LTM models, we examine memory efficiency, defined in terms of: 1) minimizing the average trials to match the sample patterns (system learning curve), and 2) minimizing the memory cell count at the end of the simulation (memory footprint). Ideally, the DMS should quickly reduce the number of trials required to match sampled patterns, while limiting the growth of memory cells (Figure 3).

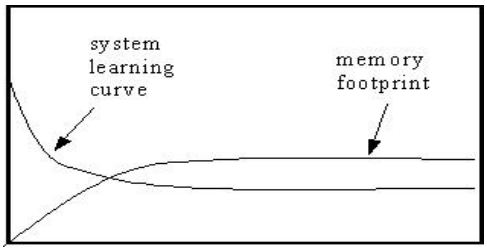


Figure 3: Model DMS behavior.

The performance of this DMS was known to depend on the values assigned to the six parameters governing the system behaviors. To fairly compare the two memory models in this DMS, we used a meta-GA to search for the best operational parameter sets for each model (Section 3). The respective parameter sets are then employed to compare the STM and LTM models. Section 4 describes the DMS simulations and provides an analysis of simulation results. Analysis of the results leads to a question of long-term stability (Section 5), where the meta-GA is again employed to search for a new DMS parameter set using new evaluation conditions. Section 6 provides insight into the effects of the STM and LTM models on genetic search, with an emphasis on the role of random genetic material.

3 Optimizing DMS Parameters Using a Meta-GA

Each evaluation for the search simulates the DMS using the operational parameters as specified by the meta-GA genes (Figure 4). The DMS is simulated for a fixed number of cycles on all nodes, where each cycle constitutes the search for a sampled pattern. Due to the stochastic nature of genetic search and the non-deterministic behavior in a multi-threaded environment, a complete DMS simulation is executed three times for each evaluation, reporting the average as the evaluation value. The evaluation function used to minimize the system learning curve and memory size is:

$$f(x) = \left[\sum_{i=0}^{i=3} (1.5 * MemCells) + avgTrials \right] /3 \quad (1)$$

The STM and LTM meta-GA searches were performed using a pattern library with 10, 64-bit strings. Two

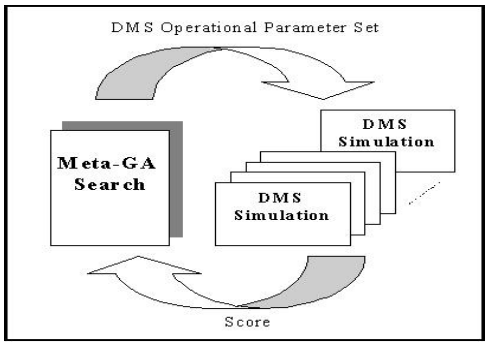


Figure 4: Meta-GA search for DMS parameter sets.

DMS nodes were concurrently simulated for 100 cycles each. For the STM model, the memory cell TTL was never reset. For the LTM model, the memory cell TTL was reset if the memory cell pattern scored higher against the current sample. The *initial* TTL was set to 100 for the simulations of both models.

3.1 Meta-GA Search Results

Table 1 shows the operational parameter sets discovered by the meta-GA searches for the two memory models. Each search was run for 8,000 trials (i.e., 24,000 simulations).[3] Both meta-GA searches discovered that a feedback rate of 2% was best for each of the two memory models. This results in a single individual being fed back to the memory cell population for each completed pattern matching cycle. The operational parameter set discovered for the LTM model specifies more agents to transport information in the DMS than does the parameter set discovered for the STM model. More agents may be required in the LTM simulations, since the agent wait time is considerably longer than the wait time specified for the STM model.

The feedback decrement values discovered for the two DMSs are similar and the direct decrement is exactly the same. However, the values discovered for the survival threshold in the two searches were dramatically different. The meta-GA search discovered a survival threshold of 50 for the DMS using a LTM model and a 12 for the STM model. The memory cells in the LTM model must be useful to the system (i.e., match a pattern sample) frequently to remain alive. A survival threshold of 50 quickly eliminates memory cells with random strings and, when combined with a direct decrement value of 3, will also eliminate memory cells with patterns that are not sampled for more than 16 cycles. The lifetime of a memory cell in the DMS using the STM model is not determined by its value to the system. Thus, a survival threshold of 12, when combined with a direct decrement of 3, allows STM cells to be exploited for ∼29 cycles.

[3]Due to the evaluation time required for each simulation, only a single meta-GA search was performed for each of the memory models proposed. Therefore, parameter sensitivity tests (i.e., varying one parameter at a time while holding the others constant) were employed to evaluate the solutions. No better solutions were discovered, indicating that the operational parameter sets at least represent local optima.

	Memory Model	
DMS Parameter	**STM**	**LTM**
Number of Agents	1	9
Agent Wait Time	338 ms	1170 ms
Feedback Rate	2%	2%
Feedback Decr.	1	0
Direct Decr.	3	3
Survival Threshold	12	50

Table 1: Parameter sets discovered by meta-GA search. Each simulation was run for 100 cycles.

4 Comparing DMS Performance for STM and LTM Models

As mentioned previously, the behavior of this DMS is stochastic in nature. Therefore, the performance of the DMS using the two memory models were compared by executing 30 independent, 100-cycle simulations, using the operational parameter sets found by the meta-GA and listed in Table 1. The results of the 30 independent simulations enabled statistical comparisons. For each simulation we measured:

1. **Final Memory Cell Count** - the total number of memory cells in the system at completion.
2. **Average Trials** - the average number of trials required to match the pattern library samples.
3. **Evaluation Value** - as given in Equation 1.
4. **Hit Rate** - the percentage of cycles where the sampled pattern was matched in the initial population (i.e., at least one memory cell used to seed the initial population matched the sample, eliminating the need for genetic search).
5. **Pattern Match Rate** - the percentage of memory cells in the system at completion that match one of the pattern library samples.
6. **Restarts/Simulation** - the number of cycles/simulation (pattern library samples) where the GA experienced at least one restart (see Section 2.2) while searching for a sampled pattern.

Table 2 gives the average values and standard error of the mean (SEM) for each of these six performance

measurements. Not surprisingly, the DMS learns to recognize patterns much faster when starting with populations seeded with memory cells than when starting from random populations (i.e., no memory).

Metric	No Memory	STM	LTM
Cell count	0	87.27 (0.69)	120.33 (1.79)
Avg trials	900.78 (7.24)	177.81 (2.24)	230.22 (3.06)
Hit rate(%)	0	84.68 (0.25)	79.83 (0.27)
Match rate(%)	N/A	69.87 (0.55)	89.35 (1.29)
Restarts/Sim	0	0.06 (0.04)	2.16 (0.35)

Table 2: Average performance values and SEM (for 30 runs) using the parameter sets found by the meta-GA (Table 1). Each run equals a 100-cycle simulation.

4.1 The Value of Short-term Memory

Performance, as measured by the evaluation value, memory cell count, and average trials, is significantly better for the DMS implementing the STM model as opposed to the LTM model. This might be unexpected since traditional memory models tend to reinforce useful memory recall events (i.e., reset TTL) and delete memory cells that have not been useful for long periods of time. However, the direct feedback that allows this DMS to learn also replenishes memory cells that are of value. For example, if a memory cell contains a string that matches the pattern sampled at a given cycle, and that memory cell is used to seed the initial population for the search, the string will be duplicated in the feedback process. This propagates useful information in the DMS without resetting the TTL.

The design of this DMS does not provide for dynamically discontinuing feedback (in which case, resetting the TTL would be critical). In future work, this DMS could respond to a dynamic pattern library, making continuous feedback critical. The emergence of a dynamically maintained distribution of patterns (represented by the memory cell population) that promotes recall at all nodes in the DMS is important.

More insight into these memory models can be gained by examining the behavior of individual DMS simulations. Figure 5-a shows the trials to match each pattern sampled during a DMS simulation of 100 cycles for the STM model, as well as, the memory cell count. Figure 5-b shows the same information for a DMS simulation using the LTM model. Cycles are shown on the X-axis. The solid line shows the number of trials to discover the pattern (Y-axis on left). The dashed line shows the memory cell count (Y-axis on the right).[4]

For both simulations, the searches usually expend ∼900 trials to discover the pattern to match the sample for the first ∼15 cycles. After this learning period, the system often contains a match for the sampled pattern in the seeded initial population. Cycles where the pattern is matched in the initial population (i.e., a hit) require only 50 trials.[5] The peaks in trials indicate genetic search was required to discover the pattern (i.e., no match was found in the initial population).

Matching a sampled pattern in the initial population has a significant impact on the average trials to find a pattern. Thus, the higher average hit rate observed for the STM DMS in Table 2 results in significantly better performance than in the LTM DMS implementation.

Figure 5-a also shows that the memory cell count grows for a period of ∼40 cycles and reaches a high of ∼100 memory cells. However, in the DMS using the LTM model (Figure 5-b) the memory cell count only grows for about half as long and peaks out at a much lower number (∼70). The shorter initial growth period and smaller memory cell count peak exhibited by the DMS using the LTM model would seem to be advantageous, but actually results in worse performance when compared with the DMS using the STM model. The memory cell count in the DMS using the STM model appears to become stable after a large number of cells expire (∼40 cycles). The DMS using the LTM model never seems to become stable and the memory cell count appears to be growing somewhat toward the end of the simulation (i.e., 100 cycles).

5 Stability Analysis of the LTM Model

The trend exhibited by the DMS using the LTM model (Figure 5-b) indicates that the memory cell count would likely continue to grow in longer simulations. To test this hypothesis, DMS simulations for the two memory models were run for 300 cycles, using the same operational parameter sets (see Table 1) used to produce Figures 5-a and 5-b. Figure 5-c shows the trials to match a sample pattern at each cycle for the DMS using the STM model over 300 cycles. Not only does the memory cell count remain stable (i.e., no growth) past the initial 100 cycles, but the peaks, indicating where genetic search is needed to discover the sampled pattern, are sparse after the first ∼30 cycles. The 300-cycle simulation for the STM model is consistent with the behavior observed in 100-cycle simulations.

[4] The information on these graphs reflects the experience at a single node in the DMS system.

[5] The entire initial population is tested regardless of which of the initial patterns match.

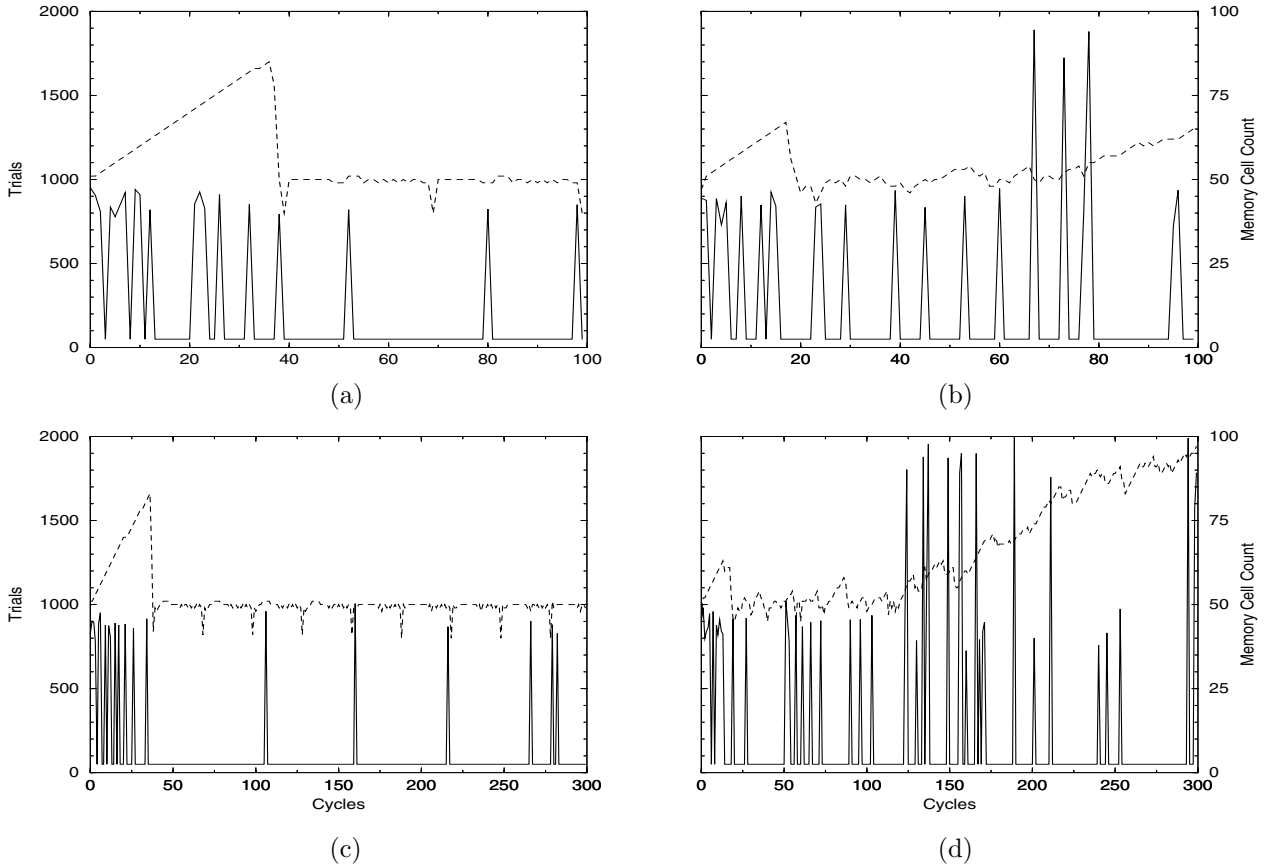


Figure 5: Trials to match sampled patterns and memory cell count at a node for a DMS simulation.

Figure 5-d shows that the memory cell count continues to grow after 100 cycles for the DMS using the LTM model, as predicted. Also, the frequency of peaks (indicating that genetic search was performed) is considerably greater than observed for the STM model simulation (Figure 5-c). This behavior indicates that the operational parameters found by the meta-GA for the DMS using the LTM model do not perform well past 100 cycles (i.e., termination point of simulations during the meta-GA search evaluations).

DMS Parameter	Memory Model		
	STM-100	**LTM-100**	**LTM-300**
Number of Agents	1	9	5
Agent Wait Time	338 ms	1170 ms	1880 ms
Feedback Rate	2%	2%	2%
Feedback Decr.	1	0	0
Direct Decr.	3	3	6
Survival Threshold	12	50	41

Table 3: Parameter sets discovered by meta-GA. The STM-100 and LTM-100 parameters sets result from the 100 cycles/simulation (Table 1) and the LTM-300 parameters result from 300 cycles/simulation.

5.1 Meta-GA Revisited

To determine if a parameter set resulting in more stable behavior could be discovered for the DMS using the LTM model, a third meta-GA search was performed using additional cycles/simulation during evaluation. The third search was performed using the same parameters, conditions and evaluation function described in Section 3, except that each simulation was run for 300 cycles instead of the original 100 cycles.

Table 3 lists the best parameters discovered after 8,000 trials for the 300 cycles/simulation evaluations (LTM-300). Table 3 also lists the parameter sets discovered for the STM and LTM models using 100 cycles/simulation during evaluation (from Table 1). The parameter sets from Table 1 are referred to as STM-100 and LTM-100 to indicate the number of cycles/simulation used during meta-search evaluations.

The LTM-300 operational parameter values discovered by the meta-GA differed in several respects from the LTM-100 parameter set. Perhaps the most critical difference is in the parameters that affect memory cell

survival. The LTM-100 parameter set included a survival threshold of 50 and a direct decrement of 3. Thus, memory cells must be useful at least every ∼16 cycles to survive. In contrast, the LTM-300 parameter set has a survival threshold of 41 and direct decrement of 6. In this case, memory cells will only be tolerated for ∼10 cycles if they are not useful. This suggests that a smaller population of memory cells will be maintained.

To compare the performance of the DMS using the LTM-300 parameter set to that of the DMS using the STM-100 and LTM-100 parameter sets, 30 independent simulations were executed using each parameter set and the corresponding memory model. All simulations were run to 300 cycles (regardless of the cycles/simulation used during meta-search). Table 4 shows the average values and SEM for the final memory cell count, average trials to match a pattern, average evaluation value, hit rate, final match rate, and average restarts/simulation for the 30 independent runs of each of the three parameter sets.

The final memory cell count, average trials to match a pattern, evaluation value, and hit rate performance values are significantly better for the DMS simulations using the STM-100 parameter set than for the DMS simulation using either the LTM-100 or LTM-300 parameter set. The LTM-300 parameter set results in only marginally better performance than the LTM-100 parameter set, despite the extra 200 cycles/simulation considered by the meta-GA. The inability to discover a parameter set resulting in better performance for the LTM model, regardless of the number of cycles/simulation, indicates the superiority of the STM model for this DMS.

Figure 6 shows the trials to find pattern matches and the memory count using the LTM-300 parameter set. The memory cell count still fluctuates after 250 cycles. However, the smaller value for the survival threshold used in the LTM-300 parameter set than the value in the LTM-100 set does in fact result in a smaller memory cell count. The average final memory cell count is significantly smaller when the LTM-300 parameter set is used rather than the LTM-100 set (Table 4).

Metric	STM-100	LTM-100	LTM-300
Mem cells	100.50 (0.16)	190.70 (7.90)	128.03 (6.67)
Avg trials	117.08 (1.04)	205.56 (5.99)	276.20 (6.81)
Eval value	267.83 (1.08)	491.61 (12.95)	468.24 (15.12)
Hit rate	91.98 (0.10)	86.06 (0.38)	76.95 (0.44)
Match rate	60.50 (0.12)	92.08 (0.85)	69.44 (2.82)
Restarts	0.27 (0.10)	26.40 (2.56)	22.3 (4.70)

Table 4: Averages for 30 independent runs using the STM-100, LTM-100 and LTM-300 parameter sets. All simulations are run to 300 cycles.

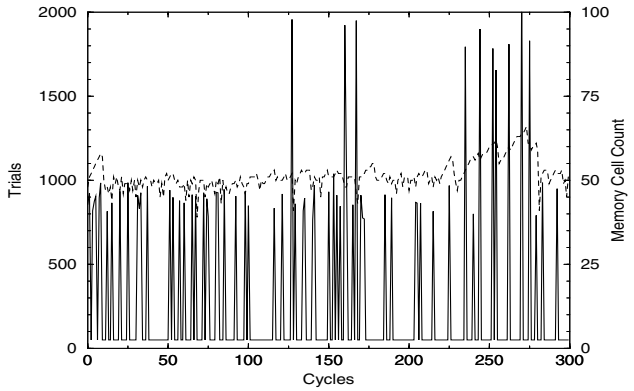


Figure 6: Trials to match patterns and memory cell count at a node for a 300 cycle DMS simulation using the LTM-300 parameter set (Table 3).

While the average final memory cell count using the LTM-300 parameters is smaller than that using the LTM-100 parameters, the average number of trials to find a matching string is significantly worse when using the LTM-300 parameter set. As previously noted, the hit rate is the most important factor in lowering the average trials required to match a sample pattern. The hit rate when using the LTM-100 parameter set is significantly higher than for the LTM-300 parameter sets. In addition, the hit rate for both of the LTM models is significantly lower than when using the STM-100 parameter set.

It is also surprising that the match rate of the final memory cell population does not correlate better with the hit rate. In fact, these two measures are inversely correlated (i.e., a high hit rate yields a lower match rate). This is due to a non-uniformly distributed memory cell population, where most of the cells match only a small fraction of the pattern library.

6 Hit Or Miss: The Role of Random Genetic Material

A side effect of the high match rate exhibited by the DMS using the LTM models (Table 4) is the occurrence of cycles where pattern samples are difficult to match. This is reflected in the large number of trials (>1200) to match a pattern (Figures 5-c, 5-d, and 6). In these instances the trials required to match a pattern are 2 to 3 times as many as when the GA uses a random initial population (i.e., no memory, Table 2). The average number of restarts/simulation for each parameter set tested is listed in Tables 2 and 4.

These events represent a restart by CHC while searching for the pattern. These restarts occur when CHC has converged on a solution that does not match the sampled pattern. This occurs when the initial population is seeded from memory cells where at least one allele in all of the seed strings disagrees with the allele at the corresponding locus of the sampled pattern. For example, if all of the seed strings have a 0 in locus 3 and the pattern sampled has a 1 in that position, the initial population will not contain the genetic material to solve the problem. CHC does not use mutation except during restarts, so the search will converge to a solution that does not match the sampled pattern. A restart will be required to find the pattern [2].

This type of "biased" seeding occurs when a large percentage of the memory cells contain strings matching patterns in the library and very few contain random strings. This condition is a result of the considerable impact of the hit rate on fitness. Memory cells containing random strings do not survive long in the DMS when the LTM model is used while memory cells with strings that match library patterns rapidly increase their representation as seen by the final match rate metric in Tables 2 and 4. In contrast, the STM model simulations contain a higher percentage of random strings in the memory cell population (i.e., significantly lower match rate). Including this random material in the initial population helps provide genetic diversity in all loci and avoids restarts more reliably.

7 Conclusions

This investigation provides an analysis of two memory models in the context of a distributed memory system. We have examined the performance impact of knowledge preservation and exploitation with respect to genetic search using CHC.

The LTM model was designed to promote and preserve high quality information, with the expectation that seeding the genetic search with this material would yield the best results. Although this model performed as expected (i.e., a high concentration of quality material is maintained), it was surprising to find that the DMS using the STM model performed significantly better with respect to average trials and memory cell count. This is due to the fact that on occasion, the LTM model causes the genetic search to be initialized with material that is not representative of the problem set (i.e., a non-uniform distribution). In these instances, the initial population is comprised of many "good" seeds, with respect to alternate pattern library samples, but not the current sample. Hence, an initial match is not available and the genetic search must discover the pattern. Even worse, there is a higher probability of a restart event, which results in performance that is significantly worse than starting with a random initial population (i.e., no memory).

The random information kept by some memory cells in the STM model actually mitigates the potential of seeding the initial population with an incorrect bias. Essentially, there exists a memory quality boundary, where highly concentrated (yet unevenly sampled) knowledge can penalize performance. This is evident from the significantly higher hit rate and better performance exhibited in simulations using the STM model. Given this behavior, we can conclude that the constant feedback mechanism in the STM model, equivalent to short term memory with reproduction, is the better of the two memory models tested for this DMS.

This research demonstrates that the exploitation of domain knowledge, or memory in this instance, can significantly expedite search performance. The STM model improved genetic search performance by a factor of five over random initialization (i.e., no memory), with respect to the number of trials needed to match a pattern. The LTM model, although performing considerably worse than STM, nevertheless demonstrated a four-fold improvement in performance over genetic search with random initialization.

References

[1] Larry Eshelman. The CHC Adaptive Search Algorithm. How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In *FOGA*, pages 265–283. Morgan Kaufmann, 1991.

[2] Larry Eshelman. Personal Communication, 2001.

[3] Stephanie Forrest, Robert Smith, Brenda Javornik, and Alan Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Journal of Evolutionary Computation*, 1(3):191–211, 1993.

[4] Sushil J. Louis and Li Gong. Augmenting Genetic Algorithms with Memory to Solve Traveling Salesman Problems. In *Joint Conference on Information Sciences*, pages 109–111. Duke University, 1997.

[5] Sushil J. Louis and Fang Zhao. Domain Knowledge for Genetic Algorithms. *International Journal of Expert Systems*, 8(3):195–212, 1995.

[6] Keith E. Mathias and L. Darrell Whitley. Changing Representations During Search: A Comparative Study of Delta Coding. *Journal of Evolutionary Computation*, 2(3):249–278, 1994.

[7] Darrell Whitley, Keith Mathias, Soraya Rana, and John Dzubera. Building Better Test Functions. In L. Eshelman, editor, *ICGA-6*. Morgan Kaufmann, 1995.

Feature Subset Selection by Estimation of Distribution Algorithms

Erick Cantú-Paz
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
cantupaz@llnl.gov

Abstract

This paper describes the application of four evolutionary algorithms to the selection of feature subsets for classification problems. Besides of a simple genetic algorithm (GA), the paper considers three estimation of distribution algorithms (EDAs): a compact GA, an extended compact GA, and the Bayesian Optimization Algorithm. The objective is to determine if the EDAs present advantages over the simple GA in terms of accuracy or speed in this problem. The experiments used a Naive Bayes classifier and public-domain and artificial data sets. All the algorithms found feature subsets that resulted in higher accuracies than using all the features. However, in contrast with other studies, we did not find evidence to support or reject the use of EDAs for this problem.

1 INTRODUCTION

In machine learning, the problem of supervised classification is concerned with using labeled examples to induce a model that classifies objects into a finite set of known classes. The examples are described by a vector of numeric or nominal features. Some of these features may be irrelevant or redundant. Avoiding irrelevant or redundant features is important because they may have a negative effect on the accuracy of the classifier. In addition, by using fewer features we may reduce the cost of acquiring the data and improve the comprehensibility of the classification model. Finding feature subsets that result in accurate classifiers can be cast as a search problem, and genetic algorithms have been used successfully to address this problem.

This paper presents experiments with a simple genetic algorithm (sGA) and three estimation of distribution algorithms (EDAs): a compact GA (cGA), an extended compact GA (ecGA), and the Bayesian Optimization Algorithm (BOA). Instead of the mutation and crossover operations of conventional GAs, EDAs use a statistical model of the individuals that survive selection to generate new individuals. EDAs are an important step toward solving the linkage problem, a fundamental obstacle to the application of simple GAs to problems with unknown relationships among variables. Numerous experimental and theoretical results show that EDAs can solve hard problems reliably and efficiently (Pelikan et al., 1999; Etxeberria & Larrañaga, 1999; Mühlenbein & Mahnig, 1999).

The objective of this study is to determine if EDAs present advantages over simple GAs in terms of accuracy or speed when applied to feature selection problems. The experiments described in this paper use public-domain and artificial data sets. The classifier was a Naive Bayes, a simple classifier that can be induced quickly, and that has been shown to have good accuracy in many problems (Kohavi & John, 1997).

Our target was to maximize the accuracy of classification. The experiments demonstrate that all the feature selection methods tried here resulted in higher accuracies than using all the features. However, in contrast with other studies, we found no evidence to support or reject the use of the advanced EDAs in this problem.

The next section briefly reviews previous applications of EAs to feature subset selection. Section 3 describes the algorithms, data sets, and the fitness evaluation method. The experimental results are presented in section 4. Section 5 concludes this paper with a summary and a discussion of future research directions.

2 FEATURE SELECTION

In a domain where objects are described by d features, there are 2^d possible feature subsets. Obviously, searching exhaustively for the best subset (using any

criteria to measure quality) is futile. One approach to deal with this problem is to preprocess the data and select features based on properties that good feature sets are presumed to have, such as orthogonality and high information content. This is known as the filter approach (John, Kohavi, & Phleger, 1994). Although it can be relatively fast, the filter approach may produce disappointing results, because it ignores completely the induction algorithm.

An alternative to preprocessing the data is the wrapper approach. The key idea is to consider the induction algorithm as a black box that can be used by a heuristic search algorithm to evaluate each candidate feature subset (John, Kohavi, & Phleger, 1994). The feature subset with the higher evaluation is selected as the final set on which to run the inducer. The resulting classifier should then be tested on data not used during the search. A major disadvantage of the wrapper approach is that it requires much more computational effort than filters.

Numerous search algorithms have been used to search for feature subsets (Jain & Zongker, 1997). Genetic algorithms are usually reported to deliver good results, but there are exceptions where simpler (and faster) algorithms result in higher accuracies on particular data sets (Jain & Zongker, 1997).

Applying GAs to the feature selection problem is straightforward: the chromosomes of the individuals contain one bit for each feature, and the value of the bit determines whether the feature will be used in the classification. Using the wrapper approach, the individuals are evaluated by training the classifiers using the feature subset indicated by the chromosome and using the resulting accuracy to calculate the fitness. Siedlecki and Sklansky (1989) were the first to describe the application of GAs in this way.

GAs have been used to search for feature subsets in conjunction with several classification methods such as neural networks (Brill et al., 1990; Brotherton & Simpson, 1995), decision trees (Bala et al., 1996), k-nearest neighbors (Kelly & Davis, 1991; Punch et al., 1993; Raymer et al., 1997; Kudo & Sklansky, 2000), rules (Vafaie & Jong, 1993), and Naive Bayes (Inza et al., 1999).

Besides selecting feature subsets, GAs can extract new features by searching for a vector of numeric coefficients that is used to transform linearly the original features (Kelly & Davis, 1991; Punch et al., 1993). In this case, a value of zero in the transformation vector is equivalent to avoiding the feature. Raymer et al. (1997) and Raymer et al. (2000) combined the linear transformation with explicit feature selection flags in the chromosomes, and reported an advantage over the pure transformation method.

The only previous application of model-building EA to select feature subsets is the work by Inza et al. (1999, 2001a, 2001b). They presented experiments with several EDAs and two sequential feature selection algorithms. Inza et al. reported that the EDAs found subsets that result in similar accuracies than the simple GA and the sequential feature selection algorithms, but the EDAs have an advantage because they need fewer generations to finish. Their algorithms are similar to those included in this study, and we use some of the same data sets.

3 METHODS

This section describes the algorithms and the data used in this study as well as the method used to evaluate the fitness.

3.1 ALGORITHMS AND DATA SETS

The simple genetic algorithm in this study uses binary strings, binary (pairwise) tournament selection without replacement, uniform crossover, and bit-wise point mutation. Simple GAs such as this have been used successfully in many applications. However, it has long been recognized that the problem-independent crossover operators used in simple GAs can disrupt groups of related variables and prevent the algorithm from reaching the global optimum, unless exponentially-sized populations are used. (Thierens (1999) gives a good description of this problem).

One approach to identify and exploit the relationships among variables is to estimate the joint distribution of the individuals that survive selection and use this model to generate new individuals. The complexity of the models has increased over time as the methods of building models from data mature and more powerful computers become available. Interested readers can consult the reviews by Pelikan et al. (1999) and Larrañaga et al. (1999).

The simplest model-building EA that was used in the experiments reported here is the compact GA (Harik, Lobo, & Goldberg, 1998). This algorithm assumes that the variables (bits) that represent the problem are independent, and therefore it models the population as a product of Bernoulli distributions. The compact GA receives its name from the compact way it represents the population: the cGA uses a vector p of length equal to the problem's length, l. Each ele-

ment of p contains the probability that a sample will take the value 1. If the Bernoulli trial is not successful the sample will be 0. All positions of p are initialized to 0.5 to simulate the usual uniform random initialization of simple GAs. New individuals are obtained by sampling consecutively from each position of p and concatenating the values obtained. The probabilities vector is updated by comparing the fitness of two individuals obtained from it. For each $p_k, k = 1, .., l$, if the fittest individual has a 1 in the k-th position, p_k is increased by $1/n$, where n is the size of the virtual population that the user wants to simulate. Likewise, if the fittest individual has a 0 in the k-th position, p_k is decreased by $1/n$. The cGA iterates until all positions in p_k contain either zero or one.

PBIL (Baluja, 1994) and the UMDA (Mühlenbein, 1998) are other examples of algorithms that use univariate models and operate on binary alphabets. They differ from the cGA in the method to update the probabilities vector.

The extended compact GA (Harik, 1999) uses a product of marginal distributions on a partition of the variables. In this model, subsets of variables can be modeled jointly, and the subsets are considered independent of other subsets. Formally, the model is $P = \prod_{i=0}^{m} P_i$, where m is the number of subsets in the partition of variables and P_i represents the distribution of the i-th subset. The distribution of a subset with k members is stored in a table with $2^k - 1$ entries. The challenge is to find a partition that models the population correctly. Harik (1999) proposed a greedy search that initially supposes that all variables are independent. The model search tries to merge all pairs of subsets and chooses the merger that minimizes a complexity measure based on information theory. The search continues until no further subsets can be merged. In contrast to the cGA, the ecGA has an explicit population that is evaluated and subject to selection at each iteration of the algorithm. The algorithm builds the model considering only those solutions that survive selection. The population is initialized randomly, and new individuals are generated by sampling consecutively from the m subset distributions.

The Bayesian Optimization Algorithm (Pelikan, Goldberg, & Cantú-Paz, 1999) models the selected individuals using a Bayesian network, which can represent dependence relations among an arbitrary number of variables. Independently, Etxeberria and Larrañaga (1999) and Mühlenbein and Mahnig (1999) introduced similar algorithms. The BOA uses a greedy search to optimize the Bayesian Dirichlet metric, a measure of how well the network represents the data (the BOA could use other metrics). The user specifies the maximum number of incoming edges to any node of the network. This number corresponds to the highest degree of interaction assumed among the variables of the problem. As the ecGA, the BOA builds the model considering only the solutions that survived selection. New individuals are generated by sampling from the network. The main difference between the ecGA and the BOA is the model that they use to represent the survivors.

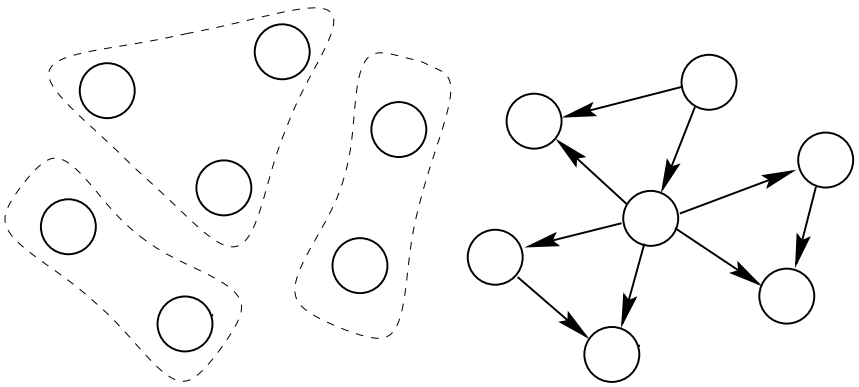
Figure 1 illustrates the different models used by the ecGA and the BOA. The ecGA cannot represent individual relationships among the variables in a subset.

The classifier induced in the experiments was a Naive Bayes (NB). This classifier was chosen for its speed and simplicity, but the evolutionary wrapper method can be used with any other supervised classifiers, as mentioned in the previous section. In the NB, the probabilities for nominal features were estimated from the data using maximum likelihood estimation (their observed frequencies in the data) and applying the Laplace correction. Numeric features were assumed to have a normal distribution. Missing values in the data were skipped.

The experiments used the C++ implementations of the ecGA (Lobo & Harik, 1999) and the BOA version 1.0 (Pelikan, 1999) that are distributed by their authors on the web.[1] The ecGA code has a non-learning mode that emulates the cGA. The sGA and Naive Bayes were developed in C++. All programs were compiled with g++ version 2.96 using -O2 optimizations. The experiments were executed on a single processor of a Linux (Red Had 7.1) workstation with dual 1.5 GHz Intel Xeon processors and 512 Mb of memory. The ecGA and the BOA codes were modified to use a Mersenne Twister random number generator, which was also used in the GA and the data partitioning.

The data sets used in the experiments are described in table 1. The first four data sets are available in the UCI repository (Blake & Merz, 1998). Random21 and Redundant21 are two artificial data sets with 21 features each. The target concept of these two data sets is to define whether the first nine features are closer to (0,0,...,0) or (9,9,...,9) in Euclidean distance. The features were generated uniformly at random in the range [3,6]. All the features in Random21 are random, and the first, fifth, and ninth features are repeated four times each in Redundant21. We took the definition of Redundant21 from the paper by Inza et al. (1999).

[1] Available at `http://www-illigal.ge.uiuc.edu`



(a) ecGA (b) BOA

Figure 1: Representation of the models used in the ecGA and the BOA.

Domain	Instances	Classes	Numeric Feat.	Nominal Feat.	Missing
Ionosphere	351	2	34	–	N
Segmentation	2310	7	19	–	N
Sick Euthyroid	3163	2	7	18	Y
Soybean Large	683	19	–	35	Y
Random21	2500	2	21	–	N
Redundant21	2500	2	21	–	N

Table 1: Description of the data used in the experiments.

3.2 MEASURING FITNESS

Since we are interested in classifiers that generalize well, the fitness calculations must include some estimate of the generalization of the Naive Bayes using the candidate subsets. If enough data are available, the generalization may be estimated by dividing the training data into training and testing sets. The training set is used to find the class conditional probabilities, and the accuracy of the trained classifier on the testing set is used to calculate the fitness.

Unfortunately, the training data sets are small, so the procedure above may not be practical in our case. Instead, we estimate the generalization of the network using crossvalidation. In k-fold crossvalidation, the data D is partitioned randomly into k non-overlapping sets, $D_1, ..., D_k$. At each iteration i (from 1 to k), the network is trained with $D \backslash D_i$ and tested on D_i. Since the data are partitioned randomly, it is likely that repeated crossvalidation experiments return different results. Although there are well-known methods to deal with "noisy" fitness evaluations in EAs (Miller & Goldberg, 1996), we chose to limit the uncertainty in the accuracy estimate by repeating 10-fold crossvalidation experiments until the standard deviation of the accuracy estimate drops below 1% (or a maximum of five repetitions). This heuristic was proposed by Kohavi and John (1997) in their study of wrapper methods for feature selection, and was adopted by Inza et al. (1999). We use the accuracy estimate as our fitness function.

Even though crossvalidation is expensive computationally, the cost was not prohibitive in our case, since the data sets were relatively small and the NB classifier is very efficient. If larger data sets or other inducers were used, we would have to deal with the uncertainty in the evaluation by other means, such as increasing slightly the population size (to compensate for the noise in the evaluation) or by sampling the training data. We defer a discussion of possible performance improvements until the final section.

Our fitness measure does not include any term to bias the search toward small feature subsets. However, the algorithms found small subsets, and with some data the algorithms consistently found the smallest subsets

that describe the target concepts. This suggests that the data sets contained irrelevant or redundant features that decreased the accuracy of the Naive Bayes.

4 EXPERIMENTS

All the algorithms used populations with 1000 individuals. The GA used uniform crossover with probability 1.0, and mutation with probability $1/l$, where l was the length of the chromosomes that corresponds to the total number of features in each problem. Promising solutions were selected with pairwise binary tournaments without replacement. The cGA, ecGA, and the BOA used the default parameters provided in their distributions: the cGA and ecGA used tournaments among 16 individuals, and the BOA used truncation selection with a threshold of 50%. The algorithms were terminated after observing no improvement in the best individual over consecutive generations.

To evaluate the generalization accuracy of the feature selection methods, we used 5 iterations of 2-fold crossvalidation (5x2cv). In each iteration, the data were randomly divided in halves. One half was input to the feature selection algorithms. The final feature subset found in each experiment was used to train a final NB classifier (using the training data), which was then tested on the other half of the data. The accuracy results presented in table 2 are the average and standard deviations of the ten tests.

To determine if the differences among the algorithms were statistically significant, we used a combined F test proposed by Alpaydin (1999). Let $p_i^{(j)}$ denote the difference in the accuracy rates of two classifiers in fold j of the i-th iteration of 5x2 cv, $\bar{p} = (p_i^{(1)} + p_i^{(2)})/2$ denote the mean, and $s_i^2 = (p_i^{(1)} - \bar{p})^2 + (p_i^{(2)} - \bar{p})^2$ the variance, then

$$f = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} \left(p_i^{(j)}\right)^2}{2 \sum_{i=1}^{5} s_i^2}$$

is approximately F distributed with 10 and 5 degrees of freedom, and we rejected the null hypothesis that the two algorithms have the same error rate with 0.95 confidence if $f > 4.74$ (Alpaydin, 1999). Care was taken to ensure that all the algorithms used the same training and testing data in the two folds of the five crossvalidation experiments. The algorithms were initialized using the same set of random seeds, so they all started from the same initial populations.

Table 2 has the average accuracies obtained with each method. The best observed result in the table is highlighted in **bold** type, and those results that according to the combined F test are significantly different from the best are marked with a bullet (•). There are two immediate observations that we can make from the results. First, the feature selection algorithms result in a great improvement in accuracy over using a NB with all the features. However, this difference is not always significant (Soybean Large, Random21). Second, all the feature selection algorithms result in similar accuracy values. There is not a single statistically significant difference among the four algorithms on these data sets.

We must be careful not to take the results at face value and conclude incorrectly that the cGA and the ecGA find feature subsets that result in better accuracies than the other EAs, since the differences are small and not significant. For the same reasons, we cannot disqualify the BOA or the simple GA, which did not score highest in any data set.

In terms of the size of the final feature subsets, all the algorithms find similarly-sized subsets, which are substantially and significantly smaller than the original set of features (see table 3). It is interesting to note that the cGA and ecGA always found subsets with the nine target features for the Redundant21 data.

Table 4 shows the mean number of generations until termination. The BOA finishes sooner than the other algorithms on most data sets, but the differences are not significant, except for one case. This observation, along with the experimental results of accuracy and feature subset size, suggests that the EDAs do not offer an advantage over the simple GA for the feature selection problems that we considered.[2]

[2] In preliminary experiments, the simple GA used a population with 100 individuals and one-point crossover (which is not particularly suitable for this problem where the ordering of the bits in the chromosome is irrelevant). The algorithms were terminated after 50 generations, although we did not observe much improvements after 10–20 generations. The cGA, ecGA, and the BOA used a population with 1000 individuals. Larger populations were chosen because these algorithms need large samples to estimate correctly the parameters of their population models. These larger populations also confer an advantage to the EDAs over the simple GA, because the EDAs sample more solutions. However, even with this advantage, we found no evidence that the EDAs found feature subsets that resulted in better classification accuracies. Moreover, we did not find significant differences in the size of the final feature subsets found by each algorithm. The simple GA was more than 10 times faster than the EDAs (because of the extra time required by the model-building step in EDAs and presumably because of random fluctuations in the number of crossvalidations used to estimate accuracy). All this lead us to favor the simple GAs over the EDAs for feature selection problems.

Domain	All Features	sGA	cGA	ecGA	BOA
Ionosphere	83.37±2.65•	90.48±1.20	**91.50±0.79**	91.05±1.91	91.22±1.01
Segmentation	79.71±0.94•	89.24±1.03	90.38±1.12	**90.44±0.91**	88.95±0.76
Soybean Large	**85.22±5.50**	84.42±4.69	84.92±5.11	85.07±5.38	83.19±4.87
Sick Euthyroid	79.04±4.23•	95.73±0.87	95.81±0.87	**95.90±0.79**	95.82±0.87
Random21	94.02±0.86	94.87±1.30	95.01±1.34	**95.07±1.25**	94.80±1.23
Redundant21	76.89±1.32•	94.16±2.40	**95.96±0.92**	**95.96±0.92**	92.66±2.59

Table 2: Mean accuracies found (± standard deviation) in the 5x2cv experiments. The best result is in **bold** and a bullet (•) denotes a result that is significantly different from the best result with 95% confidence.

Some of the results presented here agree with the conclusions of Inza et al. (1999) and Inza et al. (2001a), but some results and conclusions differ in important ways. In agreement with the results presented above, Inza et al. did not find statistically significant differences between the accuracy of their EDA and other genetic and sequential feature selection methods (using the same combined F test used here). However, they detected that the sGA needed significantly more generations to end than the EDAs in almost all the data sets they considered. This result suggests an advantage of EDAs over the sGA and the other feature selection methods they tried.

The disagreement of our results may be due to differences in the algorithms or some details in the experimental setup. It must be emphasized that the sGA and the EDAs used in this paper are not the same that Inza et al. used. An important difference is that Inza et al. used proportional selection in their simple GA, while we used tournament selection, which can be more efficient. Another difference is that the EDA of Inza et al. that learns a Bayesian network uses a greedy search that adds edges to the graph that maximize the Bayesian Information Criterion; the BOA considers edge additions and deletions and attempts to maximize a different measure of model quality. Other small differences in our experiments may affect the results slightly. For example, the Naive Bayes used in this paper was implemented from scratch, and, while great care was taken to ensure that it conformed with the specifications of their NB, differences in floating point accuracy, compilers, and operating systems can affect the results slightly.

5 CONCLUSIONS

This paper presented experiments with four evolutionary algorithms applied to the feature selection problem. The experiments considered a Naive Bayes classifier and public-domain and artificial data sets. With these data and classifier we did not find evidence to support or reject the use of the sophisticated model-building EAs in this problem. However, taking into account the (preliminary) experiments where the simple GA with smaller populations was much faster than the other algorithms and found feature subsets of similar quality, we are inclined to recommend the sGA over the other algorithms.

There are numerous opportunities to extend this work. The results that suggest that EDAs are not advantageous for feature selection should be explored further with additional data sets and other induction algorithms. It is not clear what characteristics of the data or the classifier would require an EDA to find feature subsets that reliably result in high accuracies.

Future work should also explore methods to improve the computational efficiency of the algorithms to deal with much larger data sets. In particular, subsampling the training sets and parallelizing the fitness evaluations seem like promising alternatives. In addition, future work should explore efficient methods to deal with the noisy accuracy estimates, instead of using the expensive multiple crossvalidations that we employed. Previous work (Miller & Goldberg, 1996) indicates that small increases of the population size are sufficient to deal with noise in the fitness evaluation.

Acknowledgments

I thank Martin Pelikan for providing the graphs in figure 1 and for his comments on a draft of this paper. I also thank the anonymous reviewers for their detailed comments.

UCRL-JC-146851. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Domain	Original	sGA	cGA	ecGA	BOA
Ionosphere	34•	13 ±2.35	**11.5±2.95**	12.3±1.59	14.2±2.20
Segmentation	19•	8.3±0.94	7.8±0.63	**7.4±0.84**	8.4±1.50
Soybean Large	35•	24.8±2.57	23.4±2.45	24.6±2.45	**22.4±2.67**
Sick Euthyroid	25•	12.8±2.25	12.5±2.27	12.6±3.43	**11.5±2.41**
Random21	21•	13.5±1.50	12.3±1.49	**12.1±1.37**	14±1.56•
Redundant21	21•	9.4±0.51	**9±0**	**9±0**	10±0.81

Table 3: Mean sizes of final feature subsets (± standard deviation). The best result is in **bold** and a bullet (•) denotes a result that is significantly different from the best result with 95% confidence.

Domain	sGA	cGA	ecGA	BOA
Ionosphere	**2.7±1.25**	5.3±1.88	4.6±1.77	3.3±1.49
Segmentation	2.6±1.35	4.7±1.76•	5.3±1.25	**2±1.41**
Soybean Large	4.3±2.21	3.6±1.77	4.2±1.47	**2.2±1.47**
Sick Euthyroid	2±1.05	2.2±1.03	3.4±0.84	**1.7±1.46**
Random21	2.7±1.33	3.4±1.89	4.3±0.82	**2.3±1.49**
Redundant21	3.3±2.45	3.7±0.48	4.3±1.05	**2.6±1.63**

Table 4: Mean generations until termination (± standard deviation). The best result is in **bold** and a bullet (•) denotes a result that is significantly different from the best result with 95% confidence.

References

Alpaydin, E. (1999). Combined 5 × 2cv F test for comparing supervised classification algorithms. *Neural Computation*, *11*, 1885–1892.

Bala, J., De Jong, K., Huang, J., Vafaie, H., & Wechsler, H. (1996). Using learning to facilitate the evolution of features for recognizing visual concepts. *Evolutionary Computation*, *4*(3), 297–311.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Brill, F. Z., Brown, D. E., & Martin, W. N. (1990). *Genetic algorithms for feature selection for counterpropogation networks* (Tech. Rep. No. IPC-TR-90-004). Charlottesville: University of Virginia, Institute of Parallel Computation.

Brotherton, T. W., & Simpson, P. K. (1995). Dynamic feature set training of neural nets for classification. In McDonnell, J. R., Reynolds, R. G., & Fogel, D. B. (Eds.), *Evolutionary Programming IV* (pp. 83–94). Cambridge, MA: MIT Press.

Etxeberria, R., & Larrañaga, P. (1999). Global optimization with Bayesian networks. In *II Symposium on Artificial Intelligence (CIMAF99).* (pp. 332–339).

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

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1998). The compact genetic algorithm. In of Electrical, I., & Engineers, E. (Eds.), *Proceedings of 1998 IEEE Iternational Conference on Evolutionary Computation* (pp. 523–528). Piscataway, NJ: IEEE Service Center.

Inza, I., Larrañaga, P., Etxeberria, R., & Sierra, B. (1999). Feature subset selection by Bayesian networks based on optimization. *Artificial Intelligence*, *123*(1-2), 157–184.

Inza, I., Larrañaga, P., & Sierra, B. (2001a). Feature subset selection by Bayesian networks: a comparison with genetic and sequential algorithms. *International Journal of Approximate Reasoning*, *27*(2), 143–164.

Inza, I., Larrañaga, P., & Sierra, B. (2001b). Feature subset selection by estimation of distribution algorithms. In Larrañaga, P., & Lozano, J. A. (Eds.), *Estimation of Distribution Algo-*

rithms: A new tool for Evolutionary Computation. Kluwer Academic Publishers.

Jain, A., & Zongker, D. (1997). Feature selection: evaluation, application and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(2), 153–158.

John, G., Kohavi, R., & Phleger, K. (1994). Irrelevant features and the feature subset problem. In *Proceedings of the 11th International Conference on Machine Learning* (pp. 121–129). Morgan Kaufmann.

Kelly, J. D., & Davis, L. (1991). Hybridizing the genetic algorithm and the K nearest neighbors classification algorithm. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 377–383). San Mateo, CA: Morgan Kaufmann.

Kohavi, R., & John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*(1-2), 273–324.

Kudo, M., & Sklansky, K. (2000). Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition*, *33*(1), 25–41.

Larrañaga, P., Etxeberria, R., Lozano, J. A., & Peña, J. M. (1999). *Optimization by learning and simulation of Bayesian and Gaussian networks* (Tech Report No. EHU-KZAA-IK-4/99). Conostia-San Sebastian, Spain: University of the Basque Country.

Lobo, F. G., & Harik, G. R. (1999). *Extended compact genetic algorithm in C++* (IlliGAL Report No. 99016). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Miller, B. L., & Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131.

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

Mühlenbein, H., & Mahnig, T. (1999). FDA-A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, *7*(4), 353–376.

Pelikan, M. (1999). *A simple implementation of the bayesian optimization algorithm (BOA) in C++ (version 1.0)* (IlliGAL Report No. 99011). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1* (pp. 525–532). San Francisco, CA: Morgan Kaufmann Publishers.

Pelikan, M., Goldberg, D. E., & Lobo, F. (1999). *A survey of optimization by building and using probabilistic models* (IlliGAL Report No. 99018). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Punch, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P., & Enbody, R. (1993). Further research on feature selection and classification using genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 557–564). San Mateo, CA: Morgan Kaufmann.

Raymer, M. L., Punch, W. F., Goodman, E. D., Kuhn, L. A., & Jain, A. K. (2000). Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, *4*(2), 164–171.

Raymer, M. L., Punch, W. F., Goodman, E. D., Sanschagrin, P. C., & Kuhn, L. A. (1997). Simultaneous feature scaling and selection using a genetic algorithm. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 561–567). San Francisco: Morgan Kaufmann.

Siedlecki, W., & Sklansky, J. (1989). A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, *10*, 335–347.

Thierens, D. (1999). Scalability problems of simple genetic algorithms. *Evolutionary Computation*, *7*(4), 331–352.

Vafaie, H., & Jong, K. A. D. (1993). Robust feature selection algorithms. In *Proceedings of the International Conference on Tools with Artificial Intelligence* (pp. 356–364). IEEE Computer Society Press.

On Random Numbers and the Performance of Genetic Algorithms

Erick Cantú-Paz
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550
cantupaz@llnl.gov

Abstract

Pseudo random number generators (PRNGs) are the basic input to the stochastic selection, recombination, and mutation operations of genetic algorithms (GAs). Although it does not seem like a crucial decision, recent studies suggest that the choice of PRNG can affect the performance of GAs. The objective of this paper is to study the effect of PRNGs on a simple GA, and to identify the components that are most affected by the PRNG. The paper presents ablation experiments using two PRNGs and true random numbers from an atmospheric noise source. The experiments show that the PRNG used to initialize the population is critical, but the PRNG used as input to other operations does not affect the performance significantly. We confirmed these results with additional experiments that isolated single components of the GA. In a few cases, we obtained improved results with a poor PRNG, but we were unable to obtain improvements consistently across the test functions used or with different seeds. The results suggest that, in accordance with common practice in other fields, it is preferable to use the best PRNG available to avoid muddling the interpretation of the results.

1 INTRODUCTION

A basic component of genetic algorithms (GAs) is the pseudo-random number generator (PRNG) that provides input to the stochastic selection, recombination, and mutation operations. It is well known that the performance of GAs is greatly influenced by the solution encoding, population size, and choice of operators, and it may appear that the choice of PRNG is relatively unimportant. However, several studies show that the performance of evolutionary algorithms can be affected by the choice of PRNG. In genetic programming (GP), Daida et al. (1997, 1999) found surprising improvements (ranging from 36% to 800%) on different performance measures when a *poor* PRNG was used. Meysenburg and Foster (1999a) found similar but smaller differences in GP performance. In GAs, Meysenburg (1997) and Meysenburg and Foster (1997) found that, in very few cases, a poor PRNG resulted in modest performance improvements, but they found no evidence of better GA performance with good PRNGs. Later, Meysenburg and Foster (1999b) found additional evidence of poor PRNGs causing slightly better GA performance, and also found that good PRNGs caused worse performance in isolated cases.

Our own experiments show that small variations in the PRNG can cause large deviations in the GA's performance. Consider the example in figure 1. A simple GA is optimizing a fitness function formed by concatenating 13 copies of an 8-bit trap function (defined later). The first graph shows the average fitness value reached at the end of the experiments vs. the population size. The only difference among the four plots is the random number generator used. The two overlapping plots in the middle show the results using a good PRNG (a Mersenne Twister) and true random numbers (from an atmospheric noise source); the top and bottom plots were obtained with a *poor* PRNG seeded in two different ways. In some cases, the performance with the poor PRNG seeded with an arbitrarily chosen constant (10) is 35% better than with a good PRNG and 100% better than itself seeded with random numbers (bottom plot). We observed similar trends with 7- and 9-bit traps, but we found no significant differences using other seven test functions. The second graph shows the number of trap functions that were solved to optimality (a performance measure strongly correlated to fitness) vs. the population size.

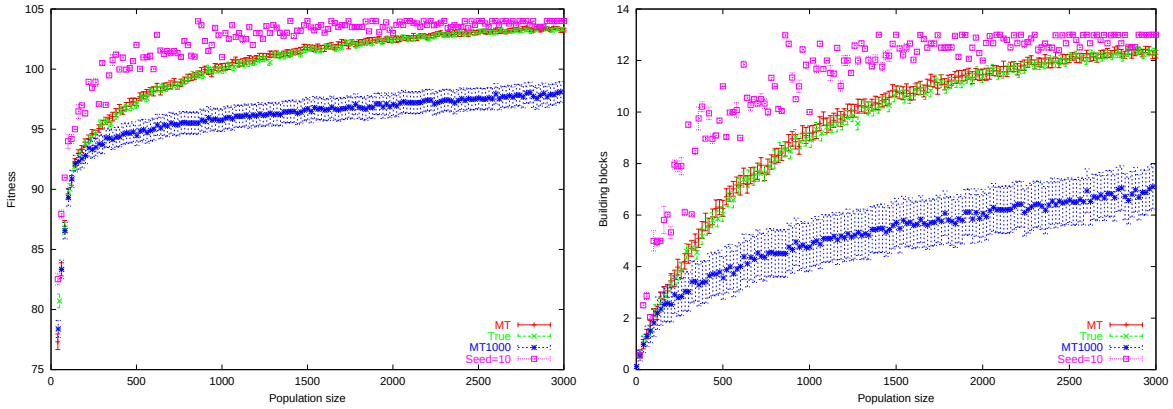


Figure 1: Example of different performance using different sources of (pseudo)random numbers. The error bars represent 95% confidence intervals. "MT" denotes experiments with a Mersenne Twister (MT), "True" denotes experiments with a source of true random numbers, "MT1000" refers to the MT with its period limited to 1000, and "Seed=10" is the limited MT initialized with the constant 10.

The objective of this paper is to continue the study of the effect of PRNGs on GAs. We used ablation experiments and simple fitness functions with known characteristics to isolate the stochastic components of the GA where the source of random numbers causes the greatest difference. In the ablation experiments, we used a "poor" PRNG as input to one of the stochastic components of the EA (say selection) while using a "good" PRNG for the rest of the algorithm. We performed additional experiments isolating single components of the GAs and compared the deviation of the algorithms to their expected behavior, which was calculated using existing models. We also performed experiments that included a source of true random numbers, but we found no difference in GA performance when compared to a "good" PRNG. We believe that this is the first time true random numbers have been used in GAs. The study was limited to simple genetic algorithms with fixed-length binary strings and popular operators.

The results of this study show that the PRNG used to initialize the population is critical to the performance of the GA, but the PRNG used as input to other GA operations does not affect the performance significantly. The experiments also show that, at least for the test functions used here, the choice of PRNG can cause large variations in performance (much larger than previously reported for GAs). Therefore, users and researchers of GAs should choose the PRNGs and their seeds carefully and report these choices appropriately, as has been advocated elsewhere (Daida et al., 1997; Daida et al., 1999).

The remainder is organized as follows: The next section describes the PRNGs and the source of true random numbers used in this study; section 3 describes the experiments and presents the results; and finally, section 4 summarizes the findings, issues recommendations, and suggests opportunities for future work.

2 (PSEUDO)RANDOM NUMBER GENERATORS

The consensus in many communities interested in stochastic simulations is to use the best PRNG available. Using the best PRNG helps to ensure that the results of a stochastic simulation are, in fact, a product of the algorithm and its inputs, and not an artifact of the PRNG.

The first PRNG used here is the Mersenne Twister (MT) (Matsumoto & Nishimura, 1998), which is considered to be one of the best PRNGs currently available (it has a period of $2^{19937}-1$ and is equidistributed in 623 dimensions). We used the implementation from the GNU Scientific Library version 0.4. In particular, this implementation uses the corrected seeding procedure recommended by the MT authors. The second PRNG used is also an MT, but its period has been artificially limited to 1000 numbers by re-seeding the generator every thousand calls with the original seed. We refer to the second PRNG as MT1000. Meysenburg and Foster (1999b) used similar generators in their experiments. In contrast to other studies that compared EA performance using numerous PRNGs, the experiments in this paper use only two generators that represent extremes in PRNG quality. This choice was motivated from the observations of Meysenburg

and Foster (1999b), where improvements in GA performance were observed only with a very poor PRNG, and no evidence of performance difference was found using other relatively good PRNGs.[1]

In addition to the two PRNGs, we use *true* random numbers obtained from an atmospheric noise source. These random numbers are available at www.random.org along with a description of the method used to create them. Briefly, a radio was tuned to a frequency where no one was transmitting, and the noise received was fed to a workstation where it was sampled as an 8-bit signal at 8KHz. The upper 7 bits of each sample were discarded, and the remaining bits were subject to a simple skew correction to ensure an even distribution of ones and zeroes.

To create our true random generator, we concatenated the four pregenerated 10Mb files available at www.random.org. These files are essentially streams of random bits that need some preprocessing before using them in a GA. The basic output from our PRNGs are uniform random numbers in $[0, 1]$. To obtain the same from the true random file, our C++ program read 4 bytes at a time into unsigned long (32 bit) integers and divided them by 2^{32}.

Unless specified otherwise, we initialized the PRNGs with 32-bit random integers obtained from the first 1Mb file from random.org. As our experiments below confirm, the initialization of the PRNGs—especially MT1000—was critical to the performance of the GA.

3 EXPERIMENTS

3.1 METHODS

The experiments used deceptive trap functions, which are used in numerous studies of genetic algorithms because they have known properties and their difficulty can be regulated easily (Deb & Goldberg, 1993). The values of the deceptive functions depend on the number, u, of bits set to one in their k-bit input substring. The fitness increases with more bits set to zero until it reaches a local optimum, but the global maximum is at the opposite extreme where all the bits in the input are set to one. The order-k traps are defined as

$$f_k(u) = \begin{cases} k - u - d & \text{if } u < k, \\ k & \text{if } u = k, \end{cases} \tag{1}$$

where d is the fitness difference of the two peaks, which in our case is always set to one. The trap functions become more difficult by increasing k and decreasing d. We varied k from 3 to 10. The fitness functions are formed by concatenating fully-deceptive trap functions and adding their individual contributions. We decided to set the length of the individuals to $l = \lceil 100/k \rceil * k$ bits (i.e., 100 bits or the smallest integer multiple of k larger than 100). For example, for the 6-bit trap problem, the individuals are $l = 102$ bits long and their fitness is calculated as $\sum_{i=0}^{16} f_6(u_{6i})$, where u_{6i} denotes the number of ones in the substring that starts at position $6i$.

The results reported here are from a simple GA with fixed-length binary encoding, pairwise tournament selection without replacement, one-point crossover with probability 1.0, and point-wise mutation with probability $1/l$. The population size for the 3,4,5,6-bit traps varied from 2 to 300 in steps of 2. For the 7-bit problem, the population size varied from 10 to 1000 in steps of 10, and for the 8,9,10-bit problems the population varied from 20 to 3000 in steps of 20. The experiments were terminated after 500 generations.

All the results were obtained repeating each experiment 100 times with different random seeds, and two-sided z tests with $\alpha = 0.05$ were used to verify if the observed means were different. The PRNGs were called each time that a random number was needed by the GA. For example, the PRNG was called once for each bit in the initial population (instead of, say, using the 32 bits returned by the PRNG to initialize 32 genes).

Our performance measure is the number of substrings that converged to the global optimal value (all ones) at the end of each run. We refer to these correct substrings as building blocks. This performance measure is adequate for the trap test functions, because the number of optimal subfunctions is a binomially distributed random variable that can be well approximated with a normal, which is what the z test assumes. This performance measure is strongly correlated with the fitness, as can be observed by comparing the two graphs in figure 1. This performance measure also allows us to calculate easily the expected behavior of the algorithm in some experiments below.

3.2 TRUE RANDOM NUMBERS

The first set of results compares the performance of the GA using the true random numbers with the MT and MT1000 PRNGs. For brevity we present only the results for 3-,4-,7-, and 10-bit problems in figure 2. The results for the 6-bit trap are similar to the 3-bit

[1] A short period is only one of the possible shortcomings of a PRNG: Correlations between consecutive samples and structural properties (such as the organization of the pseudorandom numbers in lattices) were not considered here.

problem: there are no noticeable differences, except in a few cases at relatively large population sizes, where the GA using the MT1000 performs worse. For the problems with 4,5,8,9, and 10 bit traps, the GAs using MT1000 perform noticeable worse than the GAs with true random or MT, except for small population sizes. The 7-bit problem has a similar behavior, but the transition to worse results appears at relatively large populations. These results contrast with some previous studies (Meysenburg & Foster, 1997; Meysenburg & Foster, 1999a) that showed that, in general, the performance of GAs was not adversely affected by poor PRNGs, and that sometimes poor PRNGs resulted in better results.

As mentioned in the introduction, the results in figure 1 use an 8-bit trap function. The overlapping middle plots correspond to the MT and the true random numbers. The bottom plot was generated with the MT1000 PRNG seeded with the random numbers as described in the previous section. The top plot was also generated with the MT1000 PRNG, but the seed was arbitrarily chosen to be the constant 10. Surprisingly, this poor PRNG with an arbitrary seed often outperforms all the other algorithms. We observed similar trends with the 7, and 9-bit problems, but with the other functions the results between the two MT1000 were statistically indistinguishable. While these results are intriguing, the performance with other arbitrarily chosen seeds was much worse than the *bottom* plot in figure 1. In essence, the best results were obtained by chance, and poor PRNGs do not seem to offer an advantage in general.

For all population sizes and in all problems tested, the GAs with the MT generator and the true random numbers performed equally well (there was not a single statistically significant difference). Therefore, in the following experiments we omit the results with the true random numbers.

3.3 ABLATION EXPERIMENTS

To further analyze the cause of the poor performance of GAs with MT1000, we performed ablation experiments. We started with a GA that uses MT for its four randomized components (initialization, selection, crossover, and mutation) and substituted MT1000 in each of these components at a time as specified in table 1. We also included the results where MT1000 is always used. Figure 3 has the results of this study for the 4-, 5-, 7-, and 10-bit functions. To minimize the clutter, the graphs omit the results for mutation (exp. 5), which did not differ from the results with MT.

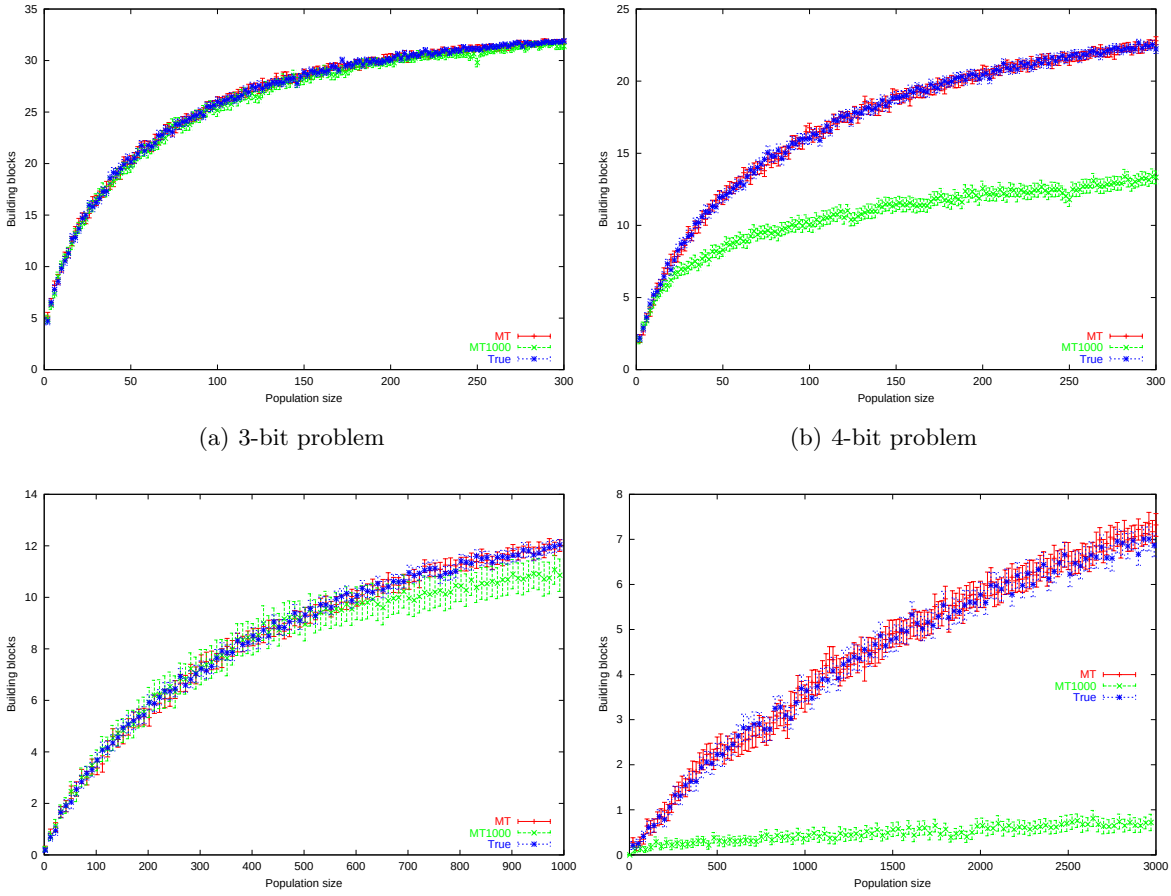
These results clearly show that the performance of the GA that used MT1000 to initialize the population (exp. 2) is strongly correlated with the performance of the GA that uses MT1000 for all its operations (exp. 6). This suggests that the initialization of the population is critical for the adequate performance of the GA, but the PRNG used in selection, crossover, or mutation seems unimportant. To provide additional support for this hypothesis, we performed an experiment where the population was initialized with MT, and the rest of the GA operations use MT1000 (experiment 7 in table 1). The results are also plotted in figure 3, and are not significantly different than those of the GA that uses MT exclusively.

Exp.	Init.	Sel.	X-over	Mut.
1	$\surd$	$\surd$	$\surd$	$\surd$
2	$\times$	$\surd$	$\surd$	$\surd$
3	$\surd$	$\times$	$\surd$	$\surd$
4	$\surd$	$\surd$	$\times$	$\surd$
5	$\surd$	$\surd$	$\surd$	$\times$
6	$\times$	$\times$	$\times$	$\times$
7	$\surd$	$\times$	$\times$	$\times$

Table 1: Ablation experiments setup. The $\surd$ and $\times$ represent the Mersenne Twister and the MT1000 PRNGs, respectively. Experiment 7 was used to verify the hypothesis that initialization was critical.

3.4 ADDITIONAL EXPERIMENTS

To try to understand why crossover and selection do not seem affected by the choice of PRNG, we performed additional experiments. To study crossover, we performed two different experiments. First, we fixed the population size to 100 and the number of generations until termination to 500. Using the 4-bit trap function and 100-bit long strings, we recorded the frequency that each possible crossover point was chosen. Ideally, we would expect that all points are chosen with the same frequency, since the probability of choosing each is uniform ($p = 1/(l-1)$). However, it is natural to expect some variability as the number of times a particular point is chosen is a random variable with a binomial distribution. Figure 4 presents the frequencies (sorted to aid in visualization) along with the expected frequency and 95% confidence intervals (calculated assuming that the binomial distribution can be approximated well with a normal). As we can see, the true random and the MT generator produce results that match our expectations, while using the MT1000 causes some crossover points to be chosen much more frequently than others. These results would suggest that the MT1000 is inadequate as input to crossover,



(a) 3-bit problem

(b) 4-bit problem

(c) 7-bit problem

(d) 10-bit problem

Figure 2: Performance of GAs using true random numbers and the MT and MT1000 PRNGs. The error bars represent 95% confidence intervals. The True and MT plots overlap, and are always at least as good as MT1000.

but in the ablation experiments we did not observe significant differences with good quality generators.

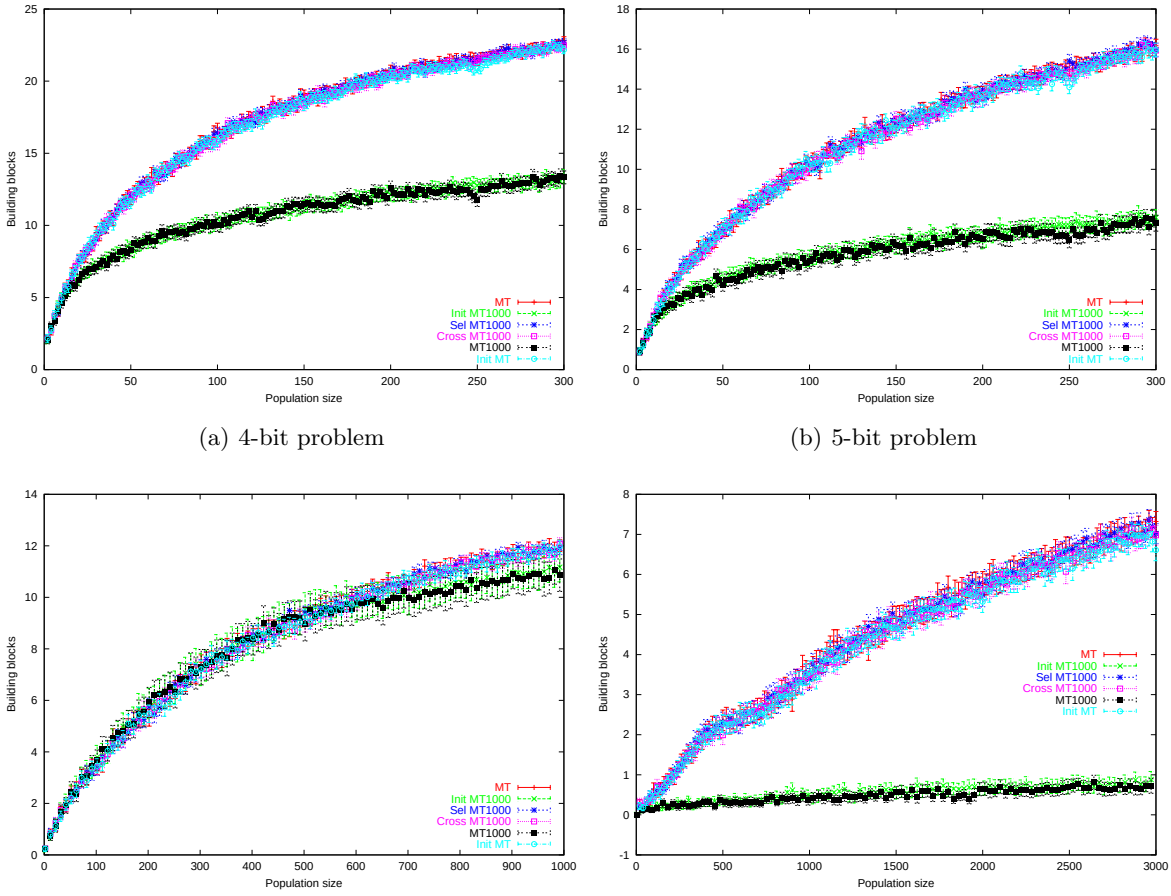
We did additional experiments recording the number of crossover operations that resulted in an offspring with at least one optimal subfunction more than each of the parents. Thierens and Goldberg (1993) call this occurrence a mixing event. As k increases, we expect fewer mixing events (all else being equal), and therefore the population was sized to 200 individuals for the functions with $k = 3, 4, 5, 6$ and to 1000 individuals for the remaining problems. The results in table 2 show that there are no significant differences in the number of mixing events using different sources of random numbers. This agrees with the ablation experiments, but it is puzzling that such a non-uniform distribution of crossover points has no apparent effect on our performance measure (or in fitness).

Separate experiments were done to investigate the effect of random inputs to selection. In particular, the following experiments verify if the expected fitness gain after selection matches the theoretical expectations. If the fitnesses are distributed normally, the mean fitness of the selected individuals can be calculated as (Mühlenbein & Schlierkamp-Voosen, 1993)

$$\mu_{\text{sel}} = \mu_{\text{orig}} + I\sigma_{\text{orig}}, \tag{2}$$

where μ_{orig} and μ_{sel} represent the mean fitness of the population before and after selection, I is the selection intensity, which in the case of pairwise tournaments is 0.5642 (Miller & Goldberg, 1995; Bäck, 1995), and σ_{orig} is the standard deviation of the fitness of the population before selection.

For these experiments, the GA used populations of 10000 individuals of length $100 * k$. We chose such a large population to ensure that MT1000 generator would cycle, and we used longer individuals to ap-



(a) 4-bit problem

(b) 5-bit problem

(c) 7-bit problem

(d) 10-bit problem

Figure 3: Performance of GAs in the ablation study. The error bars represent 95% confidence intervals. MT1000 and Init MT1000 are the two plots that overlap at the bottom; the rest of the plots overlap at the top.

proximate the assumed normal distribution. Since the fitness functions are summations of 100 random variables, it is reasonable to assume that the fitness of the initial population is distributed normally (the fitness of the selected population is certainly not normal).

We initialized the populations using the true random numbers to avoid any bias. Selection was driven by the true random numbers as well as the MT and MT1000 PRNGs. Pairwise tournament selection was applied once to the randomly initialized populations, and statistics of the selected individuals were recorded. Table 3 shows the expected and experimental results (averaged over 100 trials) of the mean fitness before and after selection. There are no statistically significant differences between the expected fitness value and the experimental results with the three sources of (pseudo)random numbers.

Finally, we performed experiments to determine the effect of PRNGs on the initialization of the population. We measured the number of optimal subfunctions in a randomly initialized population of size 1000. Table 4 has the average of 1000 repetitions using true random numbers and the different PRNGs initialized with random numbers. In addition, the table has results for MT1000 with a seed of 10, which produced the improvements in performance in figure 1. The length of the individuals was $l = \lceil 100/k \rceil * k$, and the expected number of optimal subfunctions is $l/(k * 2^k)$. Only MT1000 seeded with 10 shows significant deviations from the expected behavior. Note that MT1000 results have a higher variance (an order of magnitude) than the other generators.

k	True	MT	MT1000
3	945.48 (21.99)	947.92 (26.07)	933.42 (23.06)
4	755.87 (25.44)	770.13 (24.50)	742.82 (23.83)
5	553.56 (25.16)	554.77 (25.38)	558.03 (25.27)
6	376.56 (31.58)	372.46 (27.61)	356.67 (27.25)
7	2528.23 (107.95)	2621.69 (102.39)	2492.4 (109.20)
8	1882.13 (127.38)	1870.27 (137.06)	1806.73 (123.02)
9	957.94 (132.21)	928.89 (128.53)	847.57 (108.75)
10	421.65 (104.56)	436.84 (109.23)	423.89 (104.86)

Table 2: Number of mixing events. The numbers in parenthesis are the standard errors.

k	Original	Expected	True	MT	MT1000
3	99.9932	105.616	105.619	105.617	105.616
4	131.295	137.523	137.512	137.525	137.526
5	168.713	175.441	175.441	175.438	175.438
6	210.885	218.001	218.007	218.008	218.010
7	256.196	263.758	263.766	263.762	263.763
8	303.484	311.354	311.556	311.548	311.549
9	351.868	360.354	360.360	360.358	360.362
10	400.981	409.9156	409.921	409.911	409.924

Table 3: Population mean before and after selection driven by different PRNGs and true random numbers.

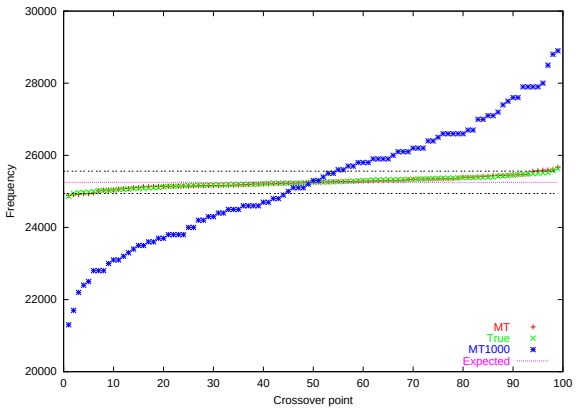


Figure 4: Frequency of choosing crossover points.

4 CONCLUSIONS

Previous studies have suggested that the choice of PRNG has a small effect on the performance of GAs. This paper presents additional experimental evidence of the effect of PRNGs on GAs, and the results suggest that the impact of the PRNG can be much more dramatic than reported previously. In agreement with other studies, we found that a poor PRNG can result in improved performance. However, this improvement is highly dependent of the seed, and we were unable to obtain good results consistently across the test functions used and different seeds. The ablation experiments suggest that the PRNG used to initialize the population is critical, while the PRNG used as input to other stochastic GA operations does not seem to affect the results. We performed additional experiments isolating individual components of the GA that seem to confirm these results. We did not observe any improvement in performance using the true random numbers over the MT generator.

The results of this study are limited to two PRNGs and to the trap functions used. Future work should apply the same experimental setup to additional functions and PRNGs. The effect of PRNGs on other evolutionary algorithms can also be studied with ablation experiments. The criticality of initialization on the performance of the GAs suggests that finding alternatives to the uniform random initialization may be beneficial.

While the choice of PRNGs seems to cause considerable fluctuations in performance, the design of reliable algorithms that consistently reach good solutions is not likely to be found in the experimentation with random seeds or different PRNGs. However, these large fluctuations require that experimenters choose their PRNGs and seeds carefully, and that these choices are reported appropriately. The results of this paper sug-

k	Expected	True	MT	MT1000 seed=random	MT1000 seed=10
3	4.25	4.2547 (0.0617)	4.2475 (0.0621)	4.2382 (0.5039)	4.653
4	1.5625	1.5668 (0.0368)	1.5632 (0.0392)	1.5635 (0.3795)	2.2
5	0.625	0.6255 (0.0236)	0.6248 (0.0259)	0.6217 (0.2462)	0.8
6	0.2656	0.2659 (0.0155)	0.2652 (0.0164)	0.2647 (0.1182)	0.34
7	0.1171	0.1175 (0.0107)	0.1172 (0.0108)	0.1149 (0.0711)	0.098
8	0.0507	0.0515 (0.0070)	0.0506 (0.0071)	0.0511 (0.0729)	0.104
9	0.0234	0.0235 (0.0045)	0.0236 (0.0049)	0.0233 (0.0295)	0.024
10	0.0097	0.0098 (0.0028)	0.0098 (0.0032)	0.0109 (0.0324)	0

Table 4: Number of optimal subfunctions in randomly initialized populations. The numbers in parenthesis are the standard errors.

gest that experimenters should use the best PRNG available to avoid "lucky" accidents that can muddle the interpretation of the results.

Acknowledgments

UCRL-JC-146850. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

I thank the anonymous reviewers for their detailed and constructive comments that helped improve the paper.

References

Bäck, T. (1995). Generalized convergence models for tournament- and (μ, λ)-selection. In Eschelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 2–8). San Francisco, CA: Morgan Kaufmann.

Daida, J., Ross, S., McClain, J., Ampy, D., & Holczer, M. (1997). Challenges with verification, repeatability, and meaningful comparisons in genetic programming. In Koza, J. R., Kalyanmoy, D., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., & Riolo, R. L. (Eds.), *Genetic Programming 97* (pp. 64–69). San Francisco, CA: Morgan Kaufmann Publishers.

Daida, J. M., Ampy, D. S., Ratanasavetavadhana, M., Li, H., & Chaudhri, O. A. (1999). Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 2* (pp. 1851–1858). San Francisco, CA: Morgan Kaufmann Publishers.

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 93–108). San Mateo, CA: Morgan Kaufmann.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, *8*(1), 3–30.

Meysenburg, M. M. (1997). *The effect of pseudo-random number generator quality on the performance of a simple genetic algorithm.* Master's thesis, University of Idaho.

Meysenburg, M. M., & Foster, J. A. (1997). The quality of pseudo-random number generators and simple genetic algorithm performance. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 276–282). San Francisco: Morgan Kaufmann.

Meysenburg, M. M., & Foster, J. A. (1999a). Random generator quality and GP performance. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 2* (pp. 1121–1126). San Francisco, CA: Morgan Kaufmann Publishers.

Meysenburg, M. M., & Foster, J. A. (1999b). Randomness and GA performance, revisited. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1* (pp. 425–432). San Francisco, CA: Morgan Kaufmann Publishers.

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, *9*(3), 193–212.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

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

Fitness Inheritance in Multi-Objective Optimization

Jian-Hung Chen
Dept. of Information Engineering
Feng Chia University
Taichung 407, Taiwan
ROC
jh.chen@ieee.org

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

Shinn-Ying Ho
Dept. of Information Engineering
Feng Chia University
Taichung 407, Taiwan
ROC
syho@fcu.edu.tw

Kumara Sastry
Dept. of General Engineering
University of Illinois
at Urbana-Champaign
Urbana, IL 61801
ksastry@uiuc.edu

Abstract

In real-world multi-objective problems, the evaluation of objective functions usually requires a large amount of computation time. Moreover, due to the curse of dimensionality, solving multi-objective problems often requires much longer computation time than solving single-objective problems. Therefore, it is essential to develop efficiency enhancement techniques for solving multi-objective problems. This paper investigates fitness inheritance as a way to speed up multi-objective genetic and evolutionary algorithms. Convergence and population-sizing models are derived and compared with experimental results in two cases: fitness inheritance without fitness sharing and fitness inheritance with fitness sharing. Results show that the number of function evaluations can be reduced with the use of fitness inheritance.

1 INTRODUCTION

For many large-scale and real-world problems, the fitness evaluation in genetic and evolutionary algorithms may be a complex simulation, model or computation. Therefore, even this subquadratic number of function evaluations is rather high. This is especially the case in solving multi-objective problems. It is not only because the number of the objectives to be evaluated is increased, but also the curse of dimensionality may increase the convergence time of genetic algorithms (GAs). As a result, it is beneficial to utilize efficiency enhancement techniques (EETs) in multi-objective GAs.

In practice EETs have improved the performance of GAs. Many real-world applications of GAs usually use EETs to improve the speed, ranged from parallel computing, distributed computing, domain-specific knowledge, or cheaper fitness functions. Recently, Sastry (2001) proposed an analytical model for analyzing and predicting behavior of single-objective GAs with EETs. However, due to the popularity of multi-objective GAs, there is a need to investigate multi-objective GAs with EETs. In this paper, one EET called fitness inheritance is modeled and optimized for greatest speedup. In fitness inheritance, an offspring sometimes inherits a fitness value from its parents rather than through function evaluations.

The objective of this paper is to model fitness inheritance and to employ this model in predicting the convergence time and population size required for the successful design of a multi-objective GA. This paper is organized in the following manner. Section 2 briefly reviews the past works on EETs and fitness sharing. Section 3 describes the bicriteria OneMax problem and fitness inheritance, and derives convergence-time and population-sizing models for multi-objective GAs with EETs, as well as the optimal proportion of inheritance, the speed-up. The experimental results on fitness inheritance with and without fitness sharing are presented in Section 4. The paper is concluded in Section 5.

2 BACKGROUND

As background information, a brief review of the fitness inheritance literature is first presented. Then, a brief summary on how to incorporate fitness inheritance in multi-objective GAs is provided. Since fitness inheritance with and without fitness sharing will be discussed in this paper, section 2.2 presents a brief summary on fitness sharing.

2.1 LITERATURE REVIEW

Smith, Dike and Stegmann (1995) proposed two ways of inheriting fitness, one by taking the average fitness of the two parents and the other by taking a weighted average of the fitness of the two parents. Their results indicated that GAs with fitness inheritance outperformed those without

inheritance in both the OneMax and an aircraft routing problem. However, theoretical analysis in this paper was limited to considering a flywheel effect that arises in the schema theorem. Zheng, Julstrom, and Cheng (1997) used fitness inheritance for the design of vector quantization codebooks. A recent study by Sastry (2001, 2001a) developed a theoretical framework for analyzing fitness inheritance, and discussed how to determine the optimal proportion of fitness inheritance and speed-up of using fitness inheritance in single-objective GAs. However, until now, there is no study on using fitness inheritance for multi-objective GAs.

2.2 FITNESS INHERITANCE

In fitness inheritance, the fitness of all the individuals in the initial population are evaluated. Thereafter, the fitness of some proportion of individuals in the subsequent population is inherited. This proportion is called the *inheritance proportion*, p_i. The remaining individuals receive evaluated fitness. If none of the individuals receive inherited fitness ($p_i = 0$), all the individuals are evaluated as usual, then no speed-up will be obtained. On the other hand, if all the individuals receive inherited fitness ($p_i = 1$), it means that none of the individuals are evaluated. Thereafter, the fitness diversity in the population will vanish rapidly and the population will premature converged, so that GAs will fail to search the global optimum. As a result, it is important to choose an optimal inheritance proportion, so that maximum speed-up will be yielded. The flowchart of multi-objective GAs with fitness inheritance is shown in figure 1.

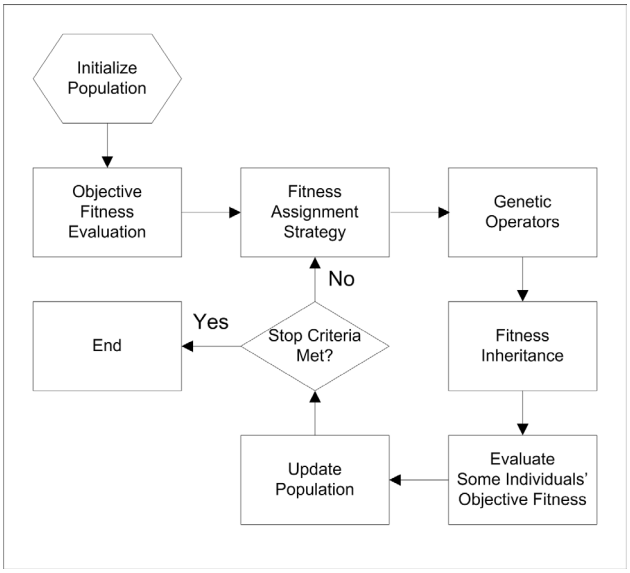


Figure 1: Fitness inheritance in multi-objective GAs.

There are several different ways to inherit fitness (objective fitness values), such as weighted-sum. For a multi-objective problem with z objective, fitness inheritance in multi-objective GAs can be defined as

$$f_z = \frac{w_1 f_{z,p1} + w_2 f_{z,p2}}{w_1 + w_2}, \qquad (1)$$

where f_z is the fitness value in objective z, w_1, w_2 are the weights for the two parents p_1, p_2, and $f_{z,p1}$, $f_{z,p2}$ is the fitness values of p_1, p_2 in objective z, respectively. In practice, fitness inheritance can be performed on all the objectives or just several objectives.

In this paper, we assume that all the objective receives inherited fitness from the parents, and the inherited fitness (objective values) is taken to be the average of the two parents. Therefore, w_1 and w_2 are set to 1.

2.3 FITNESS SHARING REVISITED

Most multi-objective problems have multiple Pareto-optimal solutions. This usually causes difficulties to any optimization algorithm in finding the global optimum solutions. In prior GA literature, there have been many niching methods on how to promote and maintain population diversity. Fitness sharing, proposed by Goldberg and Richardson (1987), may be the most widely used niching method in solving multi-modal and multi-objective problems. The basic idea of fitness sharing is to degrade the fitness of similar solutions that causes population diversity pressure. The shared fitness of an individual i is given by

$$F_{sh,i} = \frac{F_i}{m_i}, \qquad (2)$$

where F_i is the fitness of the individual, and m_i is the niche count, which defines the amount of overlap (sharing) of the individual i with the rest of the population. The niche count is calculated by summing a sharing function over all individuals of the population:

$$m_i = \sum_{j=1}^{n} sh(d_{i,j}). \qquad (3)$$

The distance $d_{i,j}$ represents the distance between individual i and individual j in the population, determined by a similarity metric. The similarity metric can be based on either phenotype or genotype similarity. If the sharing function determines that the distance is within a fixed radius σ_{sh}, it returns a value, as equation (4).

$$sh(d_{i,j}) = \begin{cases} 1 - \left(\frac{d_{i,j}}{\sigma_{sh}}\right)^{\alpha} & if\ d_{i,j} < \sigma_{sh}, \\ 0 & \text{otherwise.} \end{cases} \qquad (4)$$

The parameter α is usually set to 1. σ_{sh} is often conservatively estimated.

3 FITNESS INHERITANCE IN MULTI-OBJECTIVE OPTIMIZATION

In this section the bicriteria OneMax problem is extended from OneMax problem for analyzing multi-objective GAs

with fitness inheritance. In this section, a brief summary of fitness inheritance is also presented.

3.1 BICRITERIA ONEMAX PROBLEM

The OneMax or bit-counting problem is well-known and well-studied in the context of GAs. The OneMax problem is a bit-counting problem where fitness value of each binary string is equal to the number of one bits in it. Accordingly, the optimum binary string is an all 1s string. The simplicity of the OneMax problem makes it a prime candidate to study the effect of fitness inheritance on the performance of GAs. In order to investigate the performance of multi-objective GAs with fitness inheritance, we develop the bicriteria OneMax problem for analyzing multi-objective GAs with fitness inheritance. The bicriteria OneMax problem is defined by

$$Maximize \begin{cases} f_1(s, x_1) = l - d(s, x_1) \\ f_2(s, x_2) = l - d(s, x_2) \end{cases}, \qquad (5)$$

where string s is the string to be evaluated, x_1, and x_2 are two fixed string, the string length is l, and $d(s, x)$ is the hamming distance of two string. If the fixed string x is all 1s string, then the corresponding objective function will be the OneMax problem. The number of Pareto-optimal solutions, m, in the bicriteria OneMax problem can be calculated by

$$m = 2^{d(x_1, x_2)}. \qquad (6)$$

In this paper, unless otherwise mentioned, x_1 is all 1s string, and x_2 is all 1s string except the first four bits of x_2 is 0s.

3.2 TIME TO CONVERGENCE

In this section we derive convergence-time model for the bicriteria OneMax problem with fitness inheritance. For OneMax domain, the convergence model can be derived by using the response to selection equation (Mühlenbein and Schlierkamp-Voosen ,1993),

$$\Delta f = f_{t+1} - f_t = I\sigma_f. \qquad (7)$$

This equation was derived by calculating the difference in mean fitness of two populations using the selection intensity I, the population's fitness variance σ_f^2 at time t.

Sastry (2001) extended this model for fitness inheritance in single-objective GAs. This population-sizing model derived by Sastry is reproduced below:

$$\Delta f = f_{t+1} - f_t = I\sqrt{1 - p_i}\,\sigma_f \qquad (8)$$

Now, we can proceed to derive the convergence model for the bicriteria OneMax problem by extending equation (8). Based on the concept of fitness sharing, assumed that the population were divided into several subpopulations (niches), and each niche optimizes its own separate OneMax problem. Therefore, the optimizing process for the bicritiera OneMax problem can be regarded as optimizing several OneMax problems simultaneously. Since niches are from the same population, each niche will receive external noise from other niches. As a result, we can use the OneMax model with noisy fitness functions (Miller, 1997) to predict convergence time in the presence of external noise caused by niches. For each niche, the convergence model for the bicriteria OneMax problem can be expressed as

$$\Delta f = f_{t+1} - f_t = I\sqrt{1 - p_i}\,\frac{\sigma_f^2}{\sqrt{\sigma_f^2 + \sigma_N^2}}, \qquad (9)$$

where σ_N^2 is the noise variance from other niches.

Let M be the number of niches in the population, and

$$\rho_e = \frac{\sqrt{\sigma_f^2 + \sigma_N^2}}{\sigma_f^2}.$$

Assumed that each niche has same proportion of correct BBs, let p_t be the proportion of correct BBs in the niche at generation t. For the OneMax domain, the mean fitness at generation t equals lp_t, the fitness variance can be approximated by $lp_t(1 - p_t)$, and the noise variance from other niches can be approximated by $(M - 1)\,p_t(1 - p_t)$. The population is converged to optimal when $p_t = 1$. Equation (9) now yields

$$p_{t+1} - p_t = \frac{I}{\rho_e}\sqrt{\frac{(1 - p_i)}{l}}\sqrt{p_t(1 - p_t)}. \qquad (10)$$

Approximating the above equation with a differential equation and integrating this equation using the initial condition $p|_{t=0} = 0.5$, we get

$$p_t = \sin^2\left(\frac{\pi}{4} + \frac{It\sqrt{(1 - p_i)}}{2\rho_e\sqrt{l}}\right). \qquad (11)$$

Then we can derive an equation for convergence time, t_{conv}, by equating $p_t = 1$, and inverting equation (11),

$$t_{conv} = \frac{\pi}{2I}\sqrt{\frac{l}{(1 - p_i)}}\rho_e. \qquad (12)$$

Finally, we can yield

$$t_{conv} = \frac{\pi}{2I}\sqrt{\frac{l}{(1 - p_i)}}\sqrt{1 + \frac{M - 1}{l}}. \qquad (13)$$

If p_i is taken as 0, and M is taken as 1, then the above relation reduces to

$$t_{conv} = \frac{\pi\sqrt{l}}{2I}, \qquad (14)$$

which agrees with existing convergence-time models for the OneMax problem.

Generally, M can be set to the number of niches in the population or the number of Pareto-optimal solutions in equation (13). However, it is difficult to determine M, because niches are often overlapped in the real-world problems, and the number of niches in the population is always varied in the real runs of GAs with fitness sharing. The convergence-time model will be examined and compared with experiments in the later section.

3.3 POPULATION SIZING

Selecting a conservative population size reduces the chance of premature convergence, and it also influences the quality of the solution obtained. Therefore, it is important to appropriately size the population to incorporate the effects of fitness inheritance. For the OneMax problem, the Gambler's Ruin population-sizing model (Harik et al., 1997) can be used to determine the population-sizing model. Sastry (2001) extend this model for fitness inheritance. This population-sizing model derived by Sastry is

$$n = -\frac{2^{k-1}\ln(\psi)\sqrt{\pi}}{(1-p_i^3)}\sqrt{\sigma_f^2}\ , \qquad (15)$$

where n is the population size, k is the building block (BB) length, ψ is the failure rate, and σ_f is the variance of the noisy fitness function. For an OneMax with string length 100, k = 1, $\sigma_f^2 = 25$.

Assuming the population were divided into M niches, and each niche optimizes for its own separate OneMax problem. Similar to the population-sizing model for the bicriteria OneMax problem, we can extend this model by using the OneMax model with noisy fitness functions (Miller, 1997) to predict population-sizing in the presence of external noise caused by niches. The population model for the bicriteria OneMax problem can be written as

$$n = -\frac{2^{k-1}\ln(\psi)M\sqrt{\pi}}{(1-p_i^3)}\sqrt{\sigma_f^2+\sigma_N^2}\ , \qquad (16)$$

where σ_N^2 is the noise variance from other niches, and M is the number of niches.

The population-sizing model will be examined and compared with experiments in the later section.

3.4 OPTIMAL INHERITANCE PROPORTION AND SPEED-UP

Given a problem there should be a range of inheritance proportions that are more efficient than the others. An inappropriate inheritance proportions would not reduce the number of function evaluations. For large sized problems, Sastry's study indicates that the optimal inheritance proportion, p_i, lies between 0.54 -0.558. The total number of function evaluations required can be calculated by

$$N_{fe} = n\left[t_{conv}(1-p_i)+p_i\right]. \qquad (17)$$

From the equation (10) and equation (13), we can the predicted the total number of function evaluations required, as shown in figure 1.

The speed-up of fitness inheritance is defined as the ratio of number of function evaluations with $p_i = 0$ to the number of function evaluation at optimal p_i. From the practical view, a user usually fixes the population size and then optimizes the proportion of fitness inheritance. Therefore, the optimal proportion of fitness inheritance with a fixed number of population size can be obtained by the inverse of equation (16).

$$p_i^* = \sqrt[3]{1-\frac{\kappa}{n}}\ , \qquad (18)$$

where $\kappa = -2^{k-1}\ln(\psi)M\sqrt{\pi(\sigma_f^2+\sigma_N^2)}$. Equation (18) indicates that if the population is larger than κ, the larger the population size, the higher of inheritance proportion can be used.

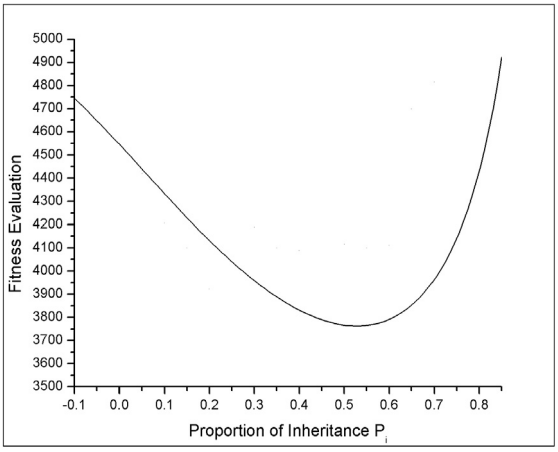


Figure 1: Total number of function evaluations predicted by equation (17) with a failure rate of 0.0001.

4 EXPERIMENTS AND RESULTS

The experiments were performed using *selectorecombinative* GAs with binary tournament selection, and uniform crossover with crossover probability of 1.0. No mutation operator is used. The sharing factor σ_{sh} is set to 50. The fitness assignment strategy we used is proposed by Ho (1999), is defined by

$$F(X) = p - q + c\ , \qquad (19)$$

where p is the number of individuals which can be dominated by the individual X, and q is the number of individuals which can dominate the individual X in the objective space. To ensure a positive fitness value, a constant c is added. Generally, the constant c can be assigned using the number of all participant individuals.

All experiments were performed 30 runs using the 100-bit bicriteria OneMax problem.

As to M in equation (13) and equation (16), considering the bicriteria OneMax problem and assuming perfect niching, M can be set to 2. Because better mixing of BBs is able to generate other Pareto-optimal solutions from x_1 and x_2. It should be an approximated lower-bound for the comparison with experimental results. However, it is noted that, in the real runs of GAs with fitness sharing, M is varied in the population. Therefore, equation (13) and equation (17) is also varied.

In order to investigate multi-objective GAs with fitness inheritance, two kind of experiments, ftiness inheritance without fitness sharing and fitness inheritance with fitness sharing, were performed and compared with analytical results. However, since multi-objective GAs without fitness sharing may lead to only some niches. Therefore, for fitness inheritance without fitness sharing, the algorithm used an external non-dominated set to store the non-dominated solutions during its search process.

4.1 FITNESS INHERITANCE WITHOUT FITNESS SHARING

The convergence time observed experimentally is compared to the above prediction for a 100-bit bicriteria OneMax problem in figure 2. Although fitness sharing was not used, the results indicate fitness inheritance is able to find all the Pareto-optimal solutions during the search process. The discrepancy between the empirical and analytical results may due to some niches disappear out of the population. Therefore, multi-objective GAs will focus the search on the remaining niches. When there is only one niche left, it lead to that all the population is optimizing an OneMax problem.

The population-sizing model is compared to the results of 100-bit OneMax problem and the results obtained for a 100-bit bicriteria OneMax problem and in figure 3. From the plot it can be easily seen that when the proportion of fitness inheritance is smaller than 0.4, our population-sizing model fits the experimental result accurately. However, when the proportion of fitness inheritance is bigger than 0.4, the experiments results get closer to the analytical results of the OneMax problem. It is because when the proportion of inheritance is higher, the diversity of population becomes lesser. So that the search was focused on the remaining niches when some niches disappeared during the search process. As a result, the convergence time of fitness inheritance without fitness sharing is varied and may be lower then the analytical results predicted by equation (13).

By using an appropriate population size and proportion of fitness inheritance and from the equation (13) and equation (16), we can the predicted the total number of function evaluations required and compared with experimental results, as shown in figure 4. The above results indicates the optimal inheritance proportion lies between 0.6 – 0.8 for fitness inheritance without fitness sharing. The speed-up is around 1.4. In other words, the number of function evaluations with inheritance is around 40% less than that without inheritance. This implies that we can get a moderate advantage by using fitness inheritance. The discrepancy between our results and Sastry's study occurs due to the disappearance of niches.

Considering the fixed population size, the speed-up is different to the speed-up obtained above. From figure 5, it can be seen that if the population size is 2000, then fitness inheritance can yield a speed-up of 3.4. The result agrees with that obtained by Sastry (2001).

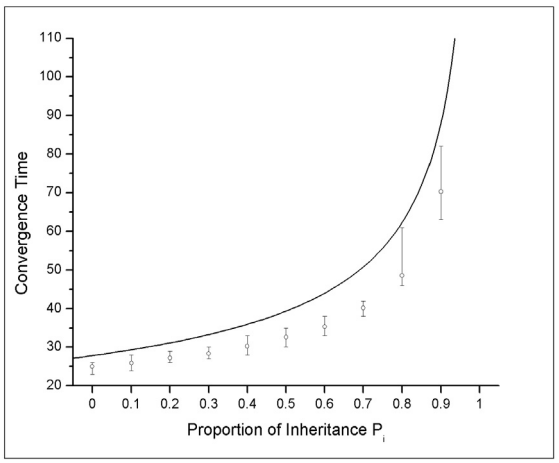


Figure 2: Convergence time for a 100-bit bicriteria OneMax problem for different proportion of inheritance predicted by equation (13) compared to experimental results.

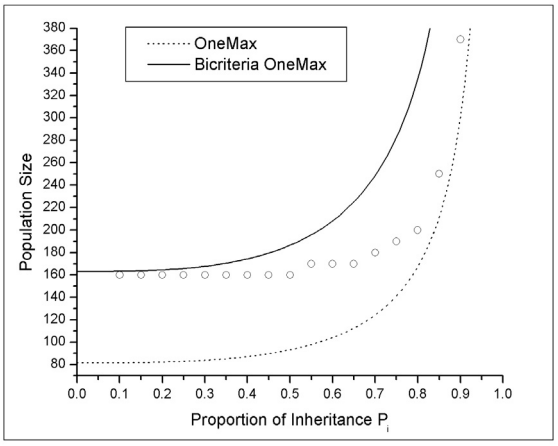


Figure 3: Verification of the population-sizing model for various inheritance proportions with empirical results. The curves are analytical results of Onemax problem and bicriteria OneMax problem, respectively. Experimental results depict the population size required for optimal convergence with failure rate of 0.0001.

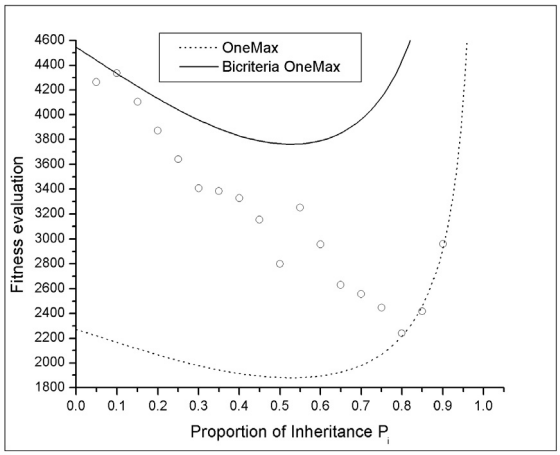


Figure 4: Total number of function evaluations predicted by equation (17) compared to experimental results. The curves are the analytical results of 100-bit Onemax problem and 100-bit bicriteria OneMax problem, respectively.

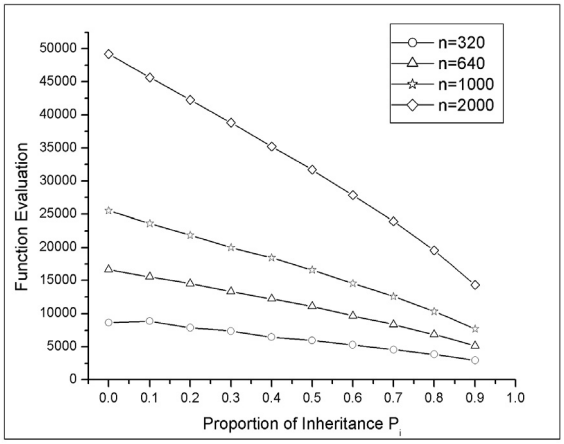


Figure 5: Total number of function evaluations for various proportion of fitness inheritance at different population sizes..

4.2 FITNESS INHERITANCE WITH FITNESS SHARING

In section 4.2, the experiments were performed using fitness inheritance with fitness sharing. The external non-dominated set was not used.

Recalling the definition of fitness sharing in section 2.3, we know that fitness sharing will degrade the fitness of similar individuals, so that these individuals will have smaller opportunity to be selected into the next generation. However, considering fitness inheritance with fitness sharing, an individual inherits fitness (objective value) from its parents. So the objective values are approximated. Then the dummy fitness is assigned according to the approximated objective values. Therefore, the dummy fitness is also approximated. Apparently, if some individuals are over-estimated and receive better fitness than their actual fitness, fitness sharing will also maintain these individuals. As a result, when fitness inheritance is used with fitness sharing, we expect that over-estimated individuals are likely to survive in the population and affect other solutions as the proportion of inheritance increased.

Figure 6 and figure 7 present the convergence model and population-sizing model observed for 100-bit bicriteria OneMax problem using fitness inheritance with fitness sharing. When the inheritance proportion is smaller than 0.7, the experimental results fit the predicted convergence model and population-sizing model. However, when the inheritance proportion is bigger than 0.8, GAs with fitness inheritance and fitness sharing cannot converge to all the Pareto-optimal solutions.

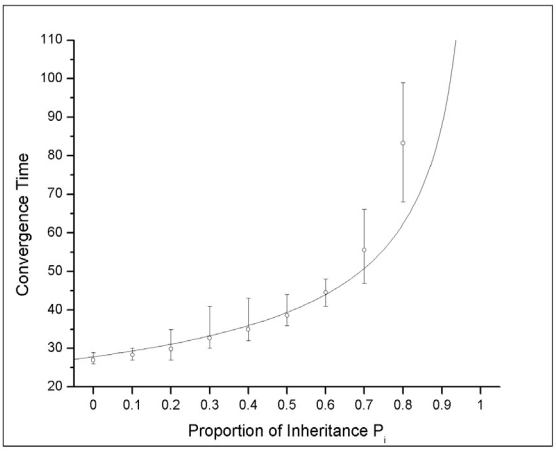


Figure 6: Convergence time for different proportion of inheritance predicted by equation (13) compared to experimental results using fitness inheritance with fitness sharing.

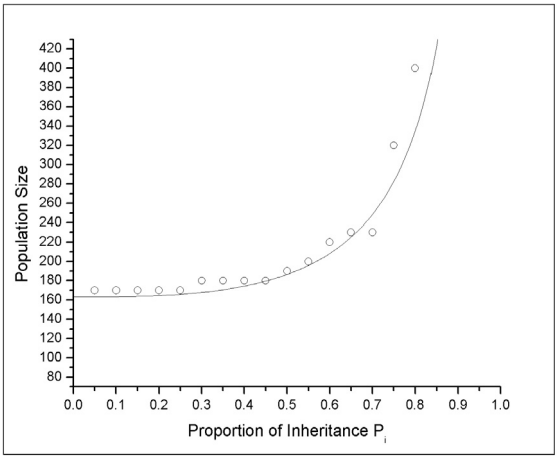


Figure 7: Verification of the population-sizing model for fitness inheritance with fitness sharing compared with empirical results. Experimental results depict the population size required for optimal convergence with failure rate of 0.0001.

Figure 8 presents the distance to Pareto front of both actual and inherited fitness for the experimental results with inheritance proportion 0.9. It indicates that the search process was divided into two phases. In this first phase, fitness inheritance proceeded well. The second phase started around the 40th generation. Some individuals were approximated to better fitness and maintained by fitness sharing. Due to the high inheritance proportion, these inferior individuals mixed with other individuals. Finally the population was filled with incorrect individuals. This phenomenon explains the discrepancy between empirical and analytical results in figure 6.

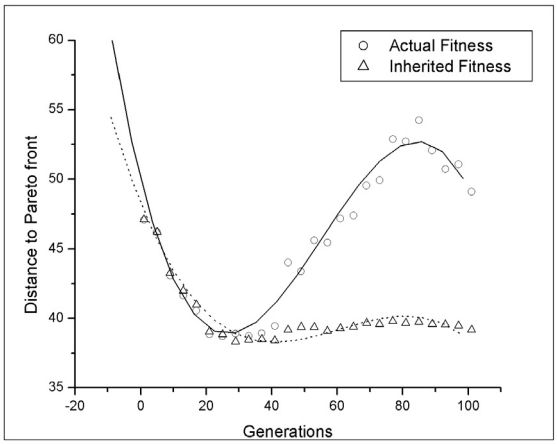


Figure 8: The distance to the Pareto front of actual fitness and inherited fitness for the experimental results with inheritance proportion 0.9. The empirical results are averaged over 30 runs.

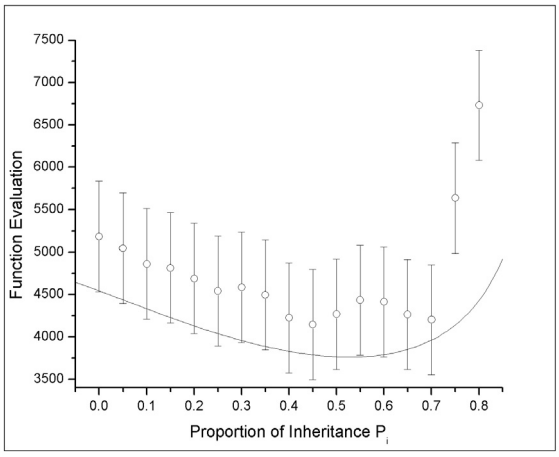


Figure 9: The distribution of function evaluations. The curve is the total number of function evaluations predicted by equation (17) for optimal convergence of a 100-bit bicriteria OneMax problem with a failure rate of 0.0001.

The predicted number of function evaluations is compared with experimental results in figure 9. The speed-up is around 1.25. The discrepancy between our results and analytical results may due to the number of niches, *M*, is varied in the real runs of GAs with fitness sharing. some inferior individuals are maintained by fitness sharing, and then mixed with other niches. Therefore, more function evaluation times are required. This may be the overhead in using GAs with fitness sharing.

In summary, the experimental results of fitness inheritance with fitness sharing indicate that the proportion of inheritance lies between 0.4 -0.5, so that incorrect niches will have lesser chance to be maintained by fitness sharing. The result is slightly different to the optimal proportion of inheritance derived by Sastry.

5 CONCLUSIONS

In this paper, we have developed a bicriteria OneMax problem and derived models for convergence-time and population-sizing. The models have been analyzed in two cases: fitness inheritance without fitness sharing and fitness inheritance with fitness sharing. In the first case, fitness inheritance yields saving on 40% in terms of the number of function evaluations. While using a fixed number of population size, fitness inheritance can yield a speed up of 3.4. In the second case, fitness inheritance yields saving to 25%.

Though the speed-up of fitness inheritance seems to be modest, it can be incorporated with parallelism, time continuation, and other efficiency enhancement techniques. In such case, a speed up of 1.25 can be important.

Further studies on using complex inheritance techniques and incorporating fitness inheritance with state-of-the-art multi-objective genetic algorithms are still remains to be done.

Acknowledgments

The authors would like to thank to Tian-Li Yu and Ying-Ping Chen for many useful comments. The first author wishes to thank the third author for his encouragement to visit the Illinois Genetic Algorithms Laboratory (IlliGAL). This paper was done during the visit.

The first author is supported by Taiwan Government Funds of Ministry of Education.

This work was partially sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-97-0050 AND F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily

representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, that National Science Foundation, or the U.S. Government.

References

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

Harik, G., Cantu-Paz, E., Goldberg, D. E., & 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*, 7-12, Piscateaway, NJ, USA: IEEE.

Ho, S.-Y. and Chang, X.-I., (1999). An efficient generalized multiobjective evolutionary algorithm. In *Proceeding of Genetic and Evolutionary Computation Conference*, 871-878. Orlando, FL, USA: Morgan Kaufmann.

Miller, B. L. (1997). *Noise, sampling and efficient genetic algorithms*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

Mühlenbein, H. & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithms: I. continuous parameter optimization. *Evolutionary Computation*, 1(1), 25-49.

Smith, R. Dike, B., & Stegmann, S. (1995). Fitness inheritance in genetic algorithms. In *Proceedings of the ACM Symposium on Applied Computing*, 345-350. New York, Y, USA: ACM.

Sastry, K. (2001). *Evaluation-relaxation schemes for genetic and evolutionary algorithms*. Master dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

Sastry, K., D. E. Goldberg & M. Pelikan (2001a). Don't evaluate, inherit. In *Proceeding of Genetic and Evolutionary Computation Conference*, 551-558. San Francisco, CA, USA: Morgan Kaufmann.

Zheng, X., Julstrom, B., & Cheng, W. (1997) Design of vector quantization codebooks using a genetic algorithm. In *Proceedings of the IEEE Conference on Evolutionary Computation*, 525-529. Picataway, NJ, USA: IEEE.

Isomorphism, Normalization, and a Genetic Algorithm for Sorting Network Optimization

Sung-Soon Choi and Byung-Ro Moon
School of Computer Science and Engineering,
Seoul National University,
Seoul, 151-742 Korea
{irranum,moon}@soar.snu.ac.kr

Abstract

In this paper, we define sorting network isomorphism and examine its relationship to graph-theoretic problems. We devise the normalization technique that exploits the functional similarities of sorting networks, which in turn helps genetic algorithms avoid too much perturbation. The sorting network isomorphism provides the basis for the normalization. In addition, we developed an effective local search heuristic for the problem. Combining the local heuristic with a genetic algorithm, we found 60-comparator sorting networks in the 16-bus problem without fixing any comparators on a single-CPU PC. This result is significantly faster and more stable than the previous study conducted with a supercomputer.

1 Introduction

A sorting network is a hardware sorting logic in which the comparisons and exchanges of data are carried out in a prescribed order. A sorting network is composed of buses and a number of homogeneous comparators, where each comparator c(a,b) performs the elementary operation that compares the a^{th} and b^{th} buses; if the values are in order, ignore them, otherwise exchange them. We call a sorting network for n inputs an n-bus sorting network. For any input sequence of an n-bus sorting network, the output sequence $y_0, y_1, \ldots, y_{n-1}$ is monotonically nondecreasing ($y_0 \leq y_1 \leq \ldots \leq y_{n-1}$). Figure 1 shows a 4-bus sorting network [c(0,1), c(2,3), c(0,2), c(1,3), c(1,2)].

Historically many scientists had made studies of 16-bus sorting networks [3] [11] [1] [21] [13]. In 1969, Green [13] 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 [14] [2] [8] [16] [10] [6] [7].

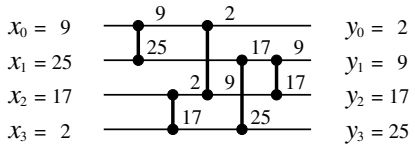


Figure 1: A 4-bus sorting network

Most of these studies initialized the networks with the first 32 comparators of Green's network to reduce the size of the problem space [14] [8]. To date Juillé [16] is the only one who attacked the problem without fixing the first 32 comparators. He found 60-comparator sorting networks with a stochastic search method called END (Evolving Non-Determinism) on a Maspar MP-2 supercomputer. Even most other studies that fixed the first 32 comparators used supercomputers [14] [8].

In [6] and [7], the authors first found 60-comparator sorting networks on a single-CPU PC with the first 32 comparators fixed. In this paper, we attack the problem without fixing any comparators. Under the single CPU environment again, we found 60-comparator sorting networks. This is thought to be possible by the following: First, we devised a process to maintain the consistency between two parent networks, which in turn helps the genetic algorithm undergo effective exploitation. Next, we designed an effective local optimization heuristic customized to the problem. Finally, we incorporated the local optimization heuristic into the genetic framework to lead a strong synergy.

The rest of this paper is organized as follows. In Section 2, we present some underlying theory related to the sorting network and our theoretical view of it. In

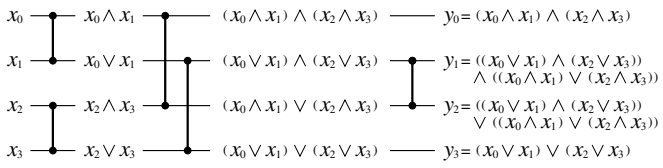


Figure 2: Formulae for the contents of buses

Section 3, we define the sorting network isomorphism and describe normalization as its application to the genetic algorithm. In Section 4, we propose our local optimization heuristic, and in Section 5 we provide a genetic algorithm combined with the local optimization heuristic. We give the experimental results in Section 6. Finally, we make our conclusions in Section 7.

2 Preliminaries

2.1 Boolean Characteristics of Sorting Networks

A valid sorting network guarantees to sort any input sequence in order. It is, however, possible to restrict the inputs to binary sequences by the following [17]:

Theorem 1 (Zero-One Principle) *If an n-bus sorting network sorts all 2^n sequences of 0's and 1's into nondecreasing order, it will sort any arbitrary sequence of n numbers into nondecreasing order.*

Then we are able to write down two outputs, min(a,b) and max(a,b), of a comparator c(a,b) as follows (note that a and b are binary numbers): $\min(a,b) = a \wedge b$ and $\max(a,b) = a \vee b$. Generalizing this, for a given sorting network, the contents of any points on a bus can be expressed in disjunctive normal form (DNF) with inputs as variables. We denote the input and output on the k^{th} bus by x_k and y_k $(0 \leq k \leq n-1)$ in an n-bus network. Figure 2 illustrates an example. In the figure, the output on 0^{th} bus, $y_0 = (x_0 \wedge x_1) \wedge (x_2 \wedge x_3)$, can be reduced to $x_0 \wedge x_1 \wedge x_2 \wedge x_3$. Similarly, y_1, y_2, and y_3 can be reduced to $(x_0 \wedge x_1 \wedge x_2) \vee (x_0 \wedge x_1 \wedge x_3) \vee \cdots \vee (x_1 \wedge x_2 \wedge x_3)$, $(x_0 \wedge x_1) \vee (x_0 \wedge x_2) \vee \cdots \vee (x_2 \wedge x_3)$, and $x_0 \vee x_1 \vee x_2 \vee x_3$, respectively, by axioms of boolean algebra.

Let σ_k be the k^{th} smallest element in the set of inputs $\{x_0, x_1, \ldots, x_{n-1}\}$ $(0 \leq k \leq n-1)$. Then, the following holds in general [17]:

$$\sigma_k = \bigvee \{ x_{i_0} \wedge x_{i_1} \wedge \cdots \wedge x_{i_{n-k-1}} | \; 0 \leq i_0 < i_1 < \cdots < i_{n-k-1} \leq n-1 \}.$$

In a valid n-bus sorting network, σ_k corresponds to the k^{th} output, i.e., $y_k = \sigma_k \; \forall \; k$.

2.2 Transforming to Valid Networks

In this section, we briefly review our previous works in [6] and [7] to transform invalid networks to valid ones.

2.2.1 O/N-pairs and Parallel Layers

The Zero-One Principle [17] says that we can prove the validity of an n-bus sorting 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 $\mathcal{T}_n$. $(\mathcal{S}_n \subseteq \mathcal{T}_n)$ We can consider an n-bus network as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Then we denote by f_χ the function corresponding to a sorting network χ.

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. 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. We call such an input bus pair (x,y) a *non-ordered pair* (*n-pair*) with respect to the network χ. 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. It is clear that $|OP(\chi)| + |NP(\chi)| = \binom{n}{2}$.

In a sorting network, a number of consecutively located independent comparators are allowed to be shuffled. We handle these interchangeable comparators as a group, since the sequence of these comparators does not affect the function of the network. We call such a group of comparators a *parallel layer*. For example, the network in Figure 1 is composed of three parallel layers.

A parallel layer strongly affects the subsequent search direction. If a considerable number of leading parallel layers have been determined, the rest may be easily constructed without redundancy by the repair heuristic of the next section. From the perspective of 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 evolved only a fixed number of leading parallel layers; this significantly reduced the computational load in [7].

2.2.2 Edit and Repair Heuristics

In [6] and [7], we devised two heuristics, edit and repair, to enhance the GAs' fine-tuning around local optima. The edit removes redundant comparators in a network. If the input bus pair of a comparator is an o-pair with respect to its previous comparators, the comparator is redundant. The repair modifies an invalid network to a valid one. In [6] and [7], the number of n-pairs of a network was used as a measure evaluating the quality of the network. The repair builds a valid network by adding a set of comparators that reduce the largest number of n-pairs. The repair consists of two operations: appending and insertion. The appending adds comparators in the rear part of a network; the insertion adds comparators in the middle of a network. See [7] for details.

3 Isomorphism and Normalization

3.1 Definition of Sorting Network Isomorphism

If we denote by $|\chi|$ the number of comparators in χ, a sorting network χ is generally represented by a sequence of comparators $\mathrm{c}_0, \mathrm{c}_1, \ldots, \mathrm{c}_{|\chi|-1}$. And each comparator c_i is represented by a pair of input bus indices (a_i,b_i). Here, the input bus indices a_i and b_i indicate only the absolute locations of the input buses of c_i. We cannot know the relationship between c_i and the input bus index system of χ, from the values of a_i and b_i.

In the previous section, we saw that an intermediate value of any point on a bus can be expressed in DNF with inputs as variables. For a comparator c_i in a network $\chi = \mathrm{c}_0, \mathrm{c}_1, \ldots, \mathrm{c}_{|\chi|-1}$ $(0 \le i \le |\chi| - 1)$, we denote by $DI(\mathrm{c}_i)$ the set of the DNFs of c_i's inputs. ($|DI(c_i)| = 2$) The sequence of $DI(c_i)$'s reflects the construction process of χ under a given input bus index system.

Let S_n be the set of all the permutations of input set $\{x_0, x_1, \ldots, x_{n-1}\}$. For a DNF $d = \bigvee_i \bigwedge_j x_{ij}$ ($x_{ij} \in \{x_0, x_1, \ldots, x_{n-1}\}$) and a permutation $p \in S_n$, we define $p(d)$ as

$$p(d) = \bigvee_i \bigwedge_j p(x_{ij}) \ .$$

Now, we define the *sorting network isomorphism* as follows:

Definition 1 *For two n-bus sorting networks* $\chi = c_0, c_1, \ldots, c_{|\chi|-1}$ *and* $\chi' = c'_0, c'_1, \ldots, c'_{|\chi'|-1}$,
χ *is* isomorphic *to* χ' $(\chi \simeq \chi')$

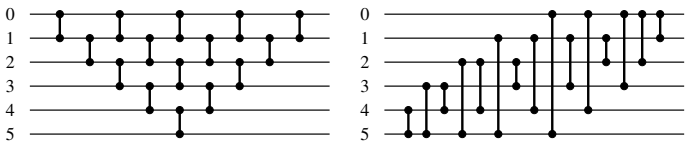


Figure 3: Two isomorphic networks

$$\iff |\chi| = |\chi'| \ \textit{and} \ \exists p \in S_n \ \textit{such that} \\ p(DI(c_i)) = DI(c'_i) \ \forall i = 0, 1, \ldots, |\chi| - 1.$$

From the definition, two isomorphic networks are constructed essentially in the same way, and their input bus indices may be different. In other words, a network can be transformed into another isomorphic network by appropriate permutation of the bus indices and comparator reconnection. Here, we should note that the comparator reconnection is more than the simple transposition of comparators' end points according to the permutation.

Figure 3 shows an example of two isomorphic networks. Between the two networks, there are a permutation $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 & 0 \end{pmatrix}$ and a nontrivial reconnection of comparators. Although they are quite different in appearance, they are the same networks in viewpoint of the construction scheme.

3.2 Validity Preserving Property Under Isomorphism

From the definition of sorting network isomorphism, given an n-bus network χ and a permutation $p \in S_n$, we can construct the network χ' isomorphic to χ. Suppose that $\chi = \mathrm{c}_0, \mathrm{c}_1, \ldots, \mathrm{c}_{|\chi|-1}$ and $\chi' = \mathrm{c}'_0, \mathrm{c}'_1, \ldots, \mathrm{c}'_{|\chi|-1}$. We choose the bus pair, as the input bus pair of c'_i, whose DNF set coincides with $p(DI(\mathrm{c}_i))$. Figure 4 shows the construction algorithm for an isomorphic network. Note that, in the algorithm, the permutation keeps changing according to the history of construction.

Generally, the validity is preserved among isomorphic networks. This property is utilized in the process of normalization in Section 3.4.

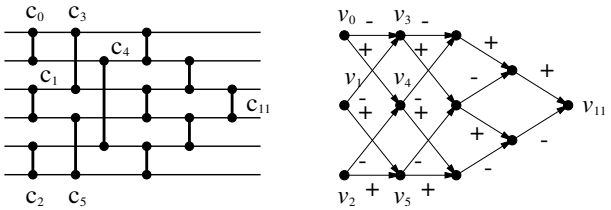
Proposition 1 *For a valid sorting network* χ*, every network isomorphic to* χ *is valid.*

Proof: Omitted by space limitation. ■

In Figure 3, the left network is valid; accordingly, the right network is also valid.

```
ConstructIsomorphicNetwork(χ, p) (const-network)
// χ : a given sorting network
//       c_0, c_1, ..., c_{|χ|-1}, c_i = (a_i, b_i)
// χ' : the isomorphic network returned
//       c'_0, c'_1, ..., c'_{|χ|-1}, c'_i = (a'_i, b'_i)
// p : a permutation ∈ S_n
{
    for i ← 0 to |χ| − 1 {
        a'_i ← min(p(a_i),p(b_i));
        b'_i ← max(p(a_i),p(b_i));
        p(a_i) ← a'_i;
        p(b_i) ← b'_i;
    }
    return χ';
}
```

Figure 4: The construction algorithm



(a) A sorting network χ (b) Its underlying graph $G(\chi)$

Figure 5: An example of an underlying graph

3.3 Underlying Graphs and Isomorphism

Given two networks χ and χ', the sorting-network isomorphism problem asks whether they are isomorphic or not. Considering the definition of sorting network isomorphism, it seems that the problem itself is rather complicated. However, we intend to look at the problem from another angle.

Sorting networks have been generally represented in such a way as in Figure 5(a). Since it draws buses across comparators in order to focus on the flows of data on buses, it is not appropriate in representing the relationship among comparators. We devised a model to better represent the relationship among comparators.

Set a vertex for each comparator. If an output of comparator c_i is fed into comparator c_j as an input, connect vertex v_i and vertex v_j by an arc e_{ij} from v_i to v_j. At this time, mark e_{ij} with "+" if it represents the maximum output of c_i. Otherwise, mark e_{ij} with "−". We have a directed graph $G = (V, E)$ corresponding to a given sorting network where V is the set of vertices and E is the set of arcs. (e.g., Figure 5(b))

For a given network χ, we call such a directed graph obtained from χ an *underlying graph* of χ and denote by $G(\chi)$. It is obvious that those graphs are acyclic and the degrees of their vertices are bounded by 4. $G(\chi)$ represents the relationships among the comparators of χ; consequently $G(\chi)$ represents the construction scheme of χ. There is strong relationship between the sorting network isomorphism and underlying graphs.

For two directed graphs $G = (V, E)$ and $G' = (V', E')$, G and G' are isomorphic if and only if there exists a bijection $f : V \mapsto V'$ such that $w(x, y) = w(f(x), f(y))$ for every ordered pair (x,y) of vertices in V, where $w(i, j)$ indicates the mark of the arc from vertex i to vertex j [18]. At this time, we denote by $G \simeq G'$. It holds, in general, that if two networks are isomorphic, then their underlying graphs are also isomorphic, and vice versa.

Proposition 2 *For two networks χ and χ',*

$$\chi \simeq \chi' \Longleftrightarrow G(\chi) \simeq G(\chi') \ .$$

Proof: Omitted by space limitation. ■

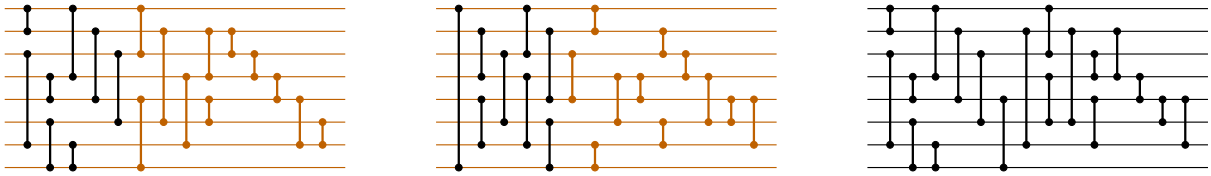
Since sorting networks can be efficiently transformed to their underlying graphs and the degree of any vertex in the underlying graphs are bounded by 4, Proposition 2 implies that the sorting network isomorphism problem is polynomially reduced to the bounded valence graph isomorphism problem.

It is an open problem whether the general graph isomorphism problem is NP-complete or not [12] [18]. However, it is known that the isomorphism of graphs of bounded valence can be tested in polynomial time [20]. So the sorting network isomorphism problem can be efficiently solved.

3.4 Normalization of Networks

If we perform crossover with two near-isomorphic solutions of significantly different shapes, the offspring will be significantly different from both parents. For example, consider Figure 6(a) and Figure 6(b). Although the subnetworks with the first eight comparators in (a) and (b) are totally different, they are isomorphic; they have the same construction schemes. If we crossover the two networks as they stand in appearance, these common attributes of the two networks are prone to be broken in the process of crossover. In a parent's point of view, the crossover is like too strong a mutation. We minimize this "visual inconsistency" before crossover.

We define the *distance* of two networks as follows:



(a) A network χ (b) A network χ' (c) χ' after normalization

Figure 6: An illustration of normalization process

Definition 2 *For two networks* $\chi = c_0, c_1, \ldots, c_{|\chi|-1}$ *and* $\chi' = c'_0, c'_1, \ldots, c'_{|\chi'|-1}$, *let* $s = max\{t | c_i = c'_i, 0 \leq i \leq t-1\}$. *The* distance *between the two networks* χ *and* χ' *is defined as*

$$d(\chi, \chi') = max\{|\chi|, |\chi'|\} - s \ .$$

It is known that a network can be reconstructed so that the distance between the network and its isomorphic one is 0.

We transform one of the parents as closely as possible to the other parent. We call such a transformation *normalization*. Although much simpler, normalization had been performed for genetic multiway graph partitioning to help maintain consistency between two parents [19].

Given a mapping of buses to minimize the distance between networks, normalization can be achieved by the construction algorithm in the previous section. In order to find the mapping, we proceed layer by layer to the right and check whether the underlying graphs, corresponding to the subnetworks (to the layer) of the two parents, are isomorphic or not. If the two graphs are not isomorphic at the k^{th} layer for the first time, we have found an isomorphism between the underlying graphs corresponding to the two subnetworks from the first layer to the $k-1^{th}$ layer. Then we match the bus indices of the corresponding source vertices (whose in-degrees are zero) of the graphs from the isomorphism, so as to generate the mapping of buses.[1] Figure 7 shows the outline of normalization.

Figure 6(c) shows the network after the network in Figure 6(b) is normalized with respect to the network in Figure 6(a), where the mapping is $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 7 & 1 & 2 & 5 & 4 & 3 & 6 \end{pmatrix}$. The actual crossover is conducted between the networks (a) and (c).

[1] [20] guarantees that such normalization can be efficiently processed in viewpoint of computation theory. Since we do not need the precise algorithm, we perform the normalization process using a type of labeling heuristic that is fast but allows errors to some degree.

```
Normalize(χ,χ')
// l, l' : numbers of parallel layers of χ and χ'
// χ_i : the i^th parallel layer of χ (0 ≤ i ≤ l − 1)
// χ'_i : the i^th parallel layer of χ' (0 ≤ i ≤ l' − 1)
{
    k ← min{l − 1, l' − 1};
    for i ← 0 to min{l − 1, l' − 1} {
        if (G(χ_0 ··· χ_i) ≇ G(χ'_0 ··· χ'_i)) {
            k ← i;
            break;
        }
    }
    if (k > 0) {
        generate the mapping p
            from G(χ_0 ··· χ_{k−1}) and G(χ'_0 ··· χ'_{k−1});
        return const-network(χ,p);
    } else {
        return χ';
    }
}
```

Figure 7: The outline of normalization

4 Local Optimization

We mentioned before that we had devised edit and repair heuristics to enhance the GA's fine-tuning around local optima in [6] and [7]. The repair heuristic is a type of greedy and constructive algorithm that modifies an invalid network to a valid one with the number of n-pairs as a measure. Although they were somewhat powerful in the version that fixes the first 32 comparators, we find that they are not enough for the version of the problem that does not fix any comparators. The problem space is incomparably huge and more powerful local optimization is desired. We designed a local search heuristic to search the problem space around a valid network after repair.

From the viewpoint of underlying graphs, the optimal sorting network problem can be considered to be the problem of finding the edges that optimally connect the vertices corresponding to given comparators — the problem of finding the optimal network topology. We consider the networks, obtained by exchanging every two input bus indices of comparators in each layer of

```
create initial population of a fixed size;
do {
    choose parent1 and parent2 from population;
    parent2 ← normalization(parent1, parent2);
    offspring ← crossover(parent1, parent2);
    mutation(offspring);
    edit(offspring);
    repair(offspring);
    local-optimization(offspring);
    replace(population, offspring);
} until (stopping condition);
return the best individual;
```

Figure 8: The outline of the hybrid genetic algorithm

χ proceeding from left to right, as neighbor networks of χ. This exchanging process is equivalent to swapping two input-output pairs of the two vertices corresponding to two comparators in the same layer in the underlying graph $G(\chi)$. Such a strategy considerably reduces the computational load of local search for the following reason.

Suppose that the number of parallel layers in χ is l, and let the parallel layers of χ be $\chi_0, \chi_1, \ldots, \chi_{l-1}$; let the functions corresponding to these parallel layers be $f_{\chi_0}, f_{\chi_1}, \ldots, f_{\chi_{l-1}}$. In general, if χ does not have any redundant comparator, the number of unsorted binary sequences steeply decreases as the parallel layers are added one by one. In other words,

$$|f_{\chi_0}(\mathcal{T}_n) - \mathcal{S}_n| \gg |f_{\chi_1}(f_{\chi_0}(\mathcal{T}_n)) - \mathcal{S}_n| \gg \cdots \gg |f_{\chi_{l-1}}(f_{\chi_{l-2}} \cdots f_{\chi_0}(\mathcal{T}_n) \cdots) - \mathcal{S}_n| .$$

Therefore, we can considerably reduce the time of evaluating the neighbors by considering only the unsorted sequences instead of all the sequences in $\mathcal{T}_n$. Such a strategy, which successively improves layers left to right, is natural in that a parallel layer strongly affects the subsequent layers [7].

5 GA Framework

We used a typical hybrid steady-state genetic algorithm. Figure 8 shows the outline of the hybrid genetic algorithm. We describe the details of our genetic algorithm in the following.

- **Encoding:** Each sorting network is represented by 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.

  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; genes in the supplemental layer are appended by repair and local optimization. This is a variant of Baldwinian hybrid GAs [22] [15] [23].

- **Initialization:** We set the population size to be 100. For each parallel layer in a chromosome, we randomly generate independent comparators. The number of independent comparators is chosen to be between a quarter and half the number of input buses. We then perform edit and repair processes to make the chromosome valid.

- **Parent 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),
  L_b : the length of the best (shortest), 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.

- **Normalization:** As mentioned in Section 3, we normalize one of the parents with respect to the other before crossover.

- **Crossover:** As mentioned, we consider only the parallel layers of the two parents in crossover. We generate five cut points. Since the lengths of the two parents are often different, we first generate five logical points and translate them into "relatively" the same positions in the parents. Few parallel layers of offsprings made in this way are usually valid.[2] We convert each layer to a valid one by removing comparators until there is no comparator that shares the same bus with another comparator in the layer.

- **Mutation:** We randomly select each comparator with a low probability (P=0.03) in each layer and change one of the input buses at random. If

[2] Here, a "valid" layer means a layer that consists of independent comparators. This usage is different from the other parts, e.g., Section 2.1, of this paper.

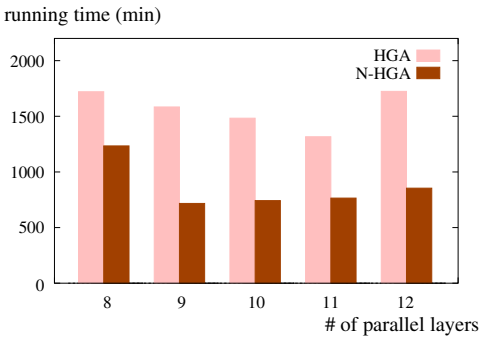


Figure 9: The average running times according to the number of parallel layers

there exists a comparator, say c, in the same layer that occupies the changed bus, we connect the comparator c to the (necessarily) absent bus after change.

- **Edit, Repair and Local Optimization:** As mentioned in Section 2, we perform the edit and repair processes to an offspring after mutation. Then we perform the local search heuristic in Section 4 to it.

- **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 [24], and showed successful results in [4].

6 Experimental Results

We evolved the population of networks using a genetic algorithm on an Intel Pentium III 866 MHz. We used a population size of 100, which is significantly smaller than Juillé's [16].

We denote by N-HGA the proposed hybrid genetic algorithm with normalization and by HGA the algorithm without the normalization. When we used 8 to 12 parallel layers, HGA found 60-comparator sorting networks in all of the 50 trials and N-HGA found 60-comparator sorting networks in all of the 100 trials. Figure 9 shows the average running times of the GAs according to the number of parallel layers. N-HGA outperformed HGA.

With 9 parallel layers, N-HGA showed the most stable performance. We conducted more experiments on the N-HGA with 9 parallel layers. Table 1 summarizes the experimental results and environments of Juillé's

Table 1: Comparison of experimental results and environments

	END [16]	N-HGA
Population size	65,536	100
Machine	Maspar MP-2 (17,000 Mips)	Pentium III 866 MHz
# of processors	4,096	1
Results	60 comparators, 2 for 3 runs	60 comparators, 50 for 50 runs
Execution time	2880 to 4320 min	346 to 1334 min (average 743 min)

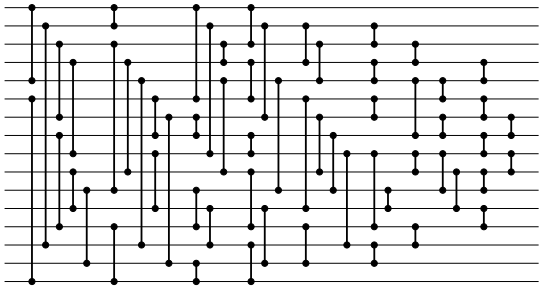


Figure 10: A 60-comparator sorting network that we found

END model [16] and the N-HGA. Juillé [16] reported that they found 60-comparator networks in two of the three runs. We found 60-comparator networks in all the 50 runs. They consumed 2 to 3 days with the Maspar MP-2 supercomputer; we consumed around half a day with a single-CPU PC. Overall, N-HGA was significantly faster and more stable than END. We present in Figure 10 one of the 60-comparator sorting networks that we found.

7 Conclusion

We defined the concept of sorting network isomorphism and examined the characteristics that isomorphic networks share. We took a graph-theoretical approach for the sorting network isomorphism by investigating its relationship with underlying graphs. We also devised the normalization technique on the basis of sorting network isomorphism and used it in the genetic algorithm, so that we could avoid more-than-necessary perturbation by crossover.

We developed an effective local search heuristic. The local search heuristic is another factor in the improvement of the performance. When the local optimization heuristic and the genetic algorithm were combined, they showed strong synergy. We found solutions with the best known quality in the 16-bus problem with a fairly small time budget. To the best of our knowledge, this is the first result that found 60-comparator sort-

ing networks without fixing any comparators under a (single-CPU) PC environment.

Although the suggested method performed impressively, we consider that there remains room for further improvement. We are currently working on the improvements for both of the local and genetic searches. We are also considering the investigation of general relationship between the isomorphism of phenotypes and the exploitation of genetic algorithms.

Acknowledgements

The authors thank Prof. Eugene M. Luks for insightful discussions about graph isomorphism. This work was supported by KOSEF through the Statistical Research Center for Complex Systems at Seoul National University (SNU) and Brain Korea 21 Project. The RIACT at SNU provided research facilities for this study.

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*, pages 457–464, 2001.

[7] S. S. Choi and B. R. Moon. A hybrid genetic search for the sorting network problem with evolving parallel layers. In *Genetic and Evolutionary Computation Conference*, pages 258–265, 2001.

[8] G. L. Drescher. Evolution of 16-number sorting networks revisited. Unpublished manuscript, 1994.

[9] C. Ebeling and O. Zaijicek. Validating VLSI circuit layout by wirelist comparison. In *Proc. of International Conference on Computer-Aided Design*, 1983.

[10] 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.

[11] 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.

[12] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[13] M. W. Green. Some improvements in nonadaptive sorting algorithms. Technical report, Stanford Research Institute, Menlo Park, California, 1969.

[14] 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.

[15] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(3):495–502, 1987.

[16] 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.

[17] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison Wesley, 1973.

[18] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.

[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] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25:42–65, 1982.

[21] G. Shapiro. In D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

[22] G. G. Simpson. The Baldwin effect. *Evolution*, 7:110–117, 1953.

[23] D. Whitley, V. Gordon, and K. Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In *International Conference on Evolutionary Computation*, pages 6–15, 1994.

[24] D. Whitley and J. Kauth. Genitor: A different genetic algorithm. In *Rocky Mountain Conf. on Artificial Intelligence*, pages 118–130, 1988.

More Effective Genetic Search for the Sorting Network Problem

Sung-Soon Choi and Byung-Ro Moon
School of Computer Science and Engineering,
Seoul National University,
Seoul, 151-742 Korea
{irranum,moon}@soar.snu.ac.kr

Abstract

In [8], the authors proposed a hybrid genetic algorithm for the optimal sorting network problem. In this paper, we propose an approach to further improve this work. For this, we designed a novel data structure to efficiently process the dominating operation in solving the problem. We also developed an effective local search heuristic based on a fast approximate measure. Combining it with genetic operators, we obtained the most stable results so far.

1 Introduction

A sorting network is composed of buses and a number of homogeneous comparators. Each comparator c(a,b) performs the elementary operation that compares the a^{th} and b^{th} buses; if their values are in order, they pass the comparator straight, otherwise they are exchanged. We call a sorting network with n inputs an n-bus sorting network. For any input sequence of an n-bus sorting network, the output sequence is monotonically non-decreasing ($y_0 \leq y_1 \leq \ldots \leq y_{n-1}$). Figure 1 shows a 4-bus sorting network with five comparators: [c(0,1), c(2,3), c(0,2), c(1,3), c(1,2)].

Usually, there are some comparators that can run simultaneously. In Figure 1, the first and second comparators can run simultaneously since they are independent. Likewise, the third and fourth comparators also can run simultaneously. Thus, the sorting can be completed in just three parallel steps.

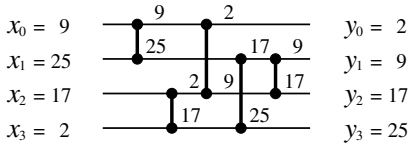


Figure 1: A 4-bus sorting network

There have been many studies of sorting networks because of their applications and rich underlying theory [17] [20] [18]. The research has two main aims [17]: to reduce the number of comparators or to reduce the number of parallel steps. We focus on minimizing the number of comparators following the convention [13] [9] [15].

This paper pursues improvement upon the authors' work on the sorting network problem [8]. We focus on the 16-bus sorting network problem. And we fix the first 32 comparators as in [8]. Historically most studies included 16-bus sorting networks [4] [11] [2] [21] [12]. In 1969, Green [12] 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 [13] [3] [9] [15] [10] [8].

Most practical studies of 16-bus sorting networks used supercomputers because of the huge computational needs [13] [9] [15]. In [7] and [8], the authors first found 60-comparator sorting networks on a single-CPU PC in a few minutes. In this study, we propose an approach to further improve this result. For this, we first designed a novel data structure in order to efficiently process the time-dominating operation in solving the problem. We also devised a fast approximate measure that evaluates the potential quality of a given network. Then we developed an effective local search heuristic based on this measure. Finally, we combined it with genetic operators to lead a strong synergy.

The rest of this paper is organized as follows. In Section 2, we present our previous works related to the sorting network problem. In Section 3, we describe the novel data structure and our local optimization heuristic, and in Section 4 we provide a genetic algo-

rithm combined with the local optimization heuristic. We give the experimental results in Section 5. Finally, we summarize the study in Section 6.

2 Preliminaries

In this section, we briefly review our previous works in [7] and [8] for further discussions.

2.1 Zero-One Principle and Networks as Functions

A valid sorting network guarantees to sort any input sequence in order. It is, however, possible to restrict the inputs to binary sequences by the following [17]:

Theorem 1 (Zero-One Principle)
If an n-bus sorting network sorts all 2^n sequences of 0's and 1's into nondecreasing order, it sorts any arbitrary sequence of n numbers into nondecreasing order.

The Zero-One Principle says that we can prove the validity of an n-bus sorting 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 and by $\mathcal{S}_n$ the set of sorted sequences from $\mathcal{T}_n$. ($\mathcal{S}_n \subseteq \mathcal{T}_n$) We can consider an n-bus network as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Then we denote by f_χ the function corresponding to a sorting network χ.

2.2 O/N-pairs and Parallel Layers

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. 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. We call such an input bus pair (x,y) a *non-ordered pair* (*n-pair*) with respect to the network χ. 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. It is clear that $|OP(\chi)| + |NP(\chi)| = \binom{n}{2}$.

In a sorting network, a number of consecutively located independent comparators are allowed to be shuffled. We handle these interchangeable comparators as a group, since the sequence of these comparators does not affect the function of the network. We call such a group of comparators a *parallel layer*. For example, the network in Figure 1 has three parallel layers.

A parallel layer strongly affects the subsequent search direction. If a considerable number of leading parallel layers have been determined, the rest may be easily constructed to the optimality by the repair heuristic of the next section. From the perspective of 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 evolved only a fixed number of leading parallel layers; this significantly reduced the computational load in [8].

2.3 Edit and Repair Heuristics

In [7] and [8], we devised two heuristics, edit and repair, to enhance the GAs' fine-tuning around local optima. The edit removes redundant comparators in a network. If the input bus pair of a comparator is an o-pair with respect to its previous comparators, the comparator is redundant. The repair modifies an invalid network to a valid one. In [7] and [8], the number of the n-pairs of a network was used as a measure evaluating the quality of the network. The repair builds a valid network by adding a set of comparators that reduce the largest number of n-pairs. The repair consists of two operations: appending and insertion. The appending adds comparators in the rear part of a network; the insertion adds comparators in the middle of a network. See [8] for details.

3 The Proposed Approach

3.1 Performance Improvement Using Function Value Tables

Rabin showed that the validity check problem of a given sorting network is co-NP-complete; it is considered that the problem is intrinsically difficult [17]. If we know the number of the n-pairs in a network, we can decide whether the network is valid or not. So it is also intrinsically difficult to find the number of the n-pairs.

Thus, it occupied most of the running time, in [8], for the edit and repair since they use the number of n-pairs as a measure of evaluating a network. They examined the output sequences obtained by feeding binary sequences into a network in order to get the number of n-pairs simultaneously with o/n-pairs, which are used in the edit and repair heuristics. Thus the comparison operation for the comparators dominated the edit and repair processes.

The local optimization algorithm described in the next section evaluates a network on the basis of the length of the edited network and the number of n-pairs.

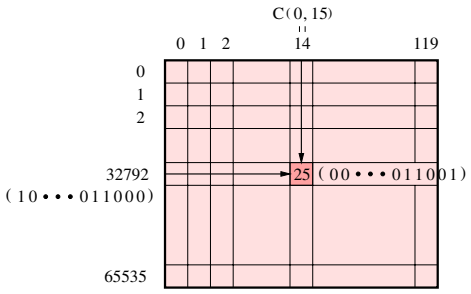


Figure 2: A function value table

The comparison operation still dominates the running time. Hence, the efficiency of the suggested approach essentially depends on how efficiently we process the comparison operation.

As mentioned above, we consider an n-bus sorting network as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Similarly, we can consider each comparator as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. We denote by $|\chi|$ the number of comparators in a sorting network χ. Suppose that an n-bus network χ consists of the comparators $c_0, c_1, \ldots, c_{|\chi|-1}$. If we denote by f_χ the function corresponding to χ and by f_{c_i} the function corresponding to c_i $(0 \le i \le |\chi| - 1)$, then, for any $t \in \mathcal{T}_n$, it holds that

$$\begin{aligned} f_\chi(t) &= (f_{c_{|\chi|-1}} \circ f_{c_{|\chi|-2}} \circ \cdots \circ f_{c_1} \circ f_{c_0})(t) \\ &= f_{c_{|\chi|-1}}(\, f_{c_{|\chi|-2}}\, (\cdots f_{c_1}(f_{c_0}(t)) \cdots)) \,. \end{aligned}$$

From the above equation, if we know the images of all binary sequences under f_{c_i} for all c_i $(0 \le i \le |\chi| - 1)$, we could get the value $f_\chi(t)$ for any $t \in \mathcal{T}_n$ in just $|\chi|$ table lookups, instead of comparing a sequence of pairs.

We prepare a matrix in advance. In the matrix, each row represents an integer in $[0, 2^n - 1]$; each integer corresponds to a binary sequence in $\mathcal{T}_n$. Each column represents an integer in $[0, \binom{n}{2} - 1]$ and corresponds to a comparator. The element m_{ij} in the matrix represents the binary sequence after applying the j^{th} comparator to the i^{th} binary sequence. In the case of the 16-bus network problem, the memory space required to create such a table is $\binom{16}{2} \times 2^{16} \times 2 = 120 \times 65536 \times 2$ bytes $<$ 16 Mbytes. Figure 2 shows the matrix for the 16-bus problem. In addition to this, we generate a table that, for each $t \in \mathcal{T}_n$, stores the pairs in t that are not in order. By using these tables, we could considerably reduce the computational time for the comparison operations.[1]

[1]Experiments showed that the computational time was reduced to less than one third by using the tables.

3.2 A Local Search Heuristic

3.2.1 An Approximate Measure for Network Quality

We mentioned before that we had devised edit and repair heuristics to enhance the GA's fine-tuning around local optima in [7] and [8]. The repair heuristic is a type of greedy and constructive algorithm that modifies an invalid network to a valid one with the number of n-pairs as a measure. In general, the perturbation by crossover and mutation considerably lowers the qualities of offspring networks. In this case, it is difficult to recover the qualities of networks only by the edit and the *greedy* and *constructive* repair heuristic. In this study, we devised a local search heuristic to alleviate this problem.

Let χ' be a neighbor of a network χ in the problem space. The eventual quality of χ' can be the quality of the network obtained by performing the edit and repair to χ'. Due to the tablization of the previous section, we can considerably reduce the time for edit and repair. Nevertheless, the repair is still a heavy-computing operation.

For a sorting network χ, let $E(\chi)$ and $R(\chi)$ be the networks obtained by performing edit and repair heuristics to χ, respectively. We get the following fact from Fact 2 in [7]:

Fact 1 *For a sorting network χ,*

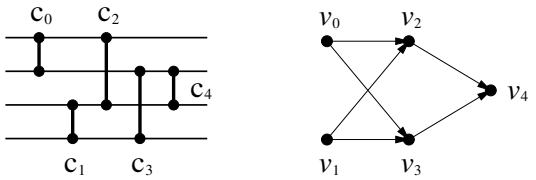
$$|R(E(\chi))| \le |E(\chi)| + |NP(\chi)| \,.$$

Proof: Omitted. ■

We denote by $\widehat{|\chi|}$ the value of $|E(\chi)|+|NP(\chi)|$ and call $\widehat{|\chi|}$ as the *potential-length upper bound* of χ. From the above fact, $\widehat{|\chi|}$ is an upper bound of the length of the network obtained by performing the edit and repair heuristics to χ. In other words, the quality of χ is bounded by $\widehat{|\chi|}$.

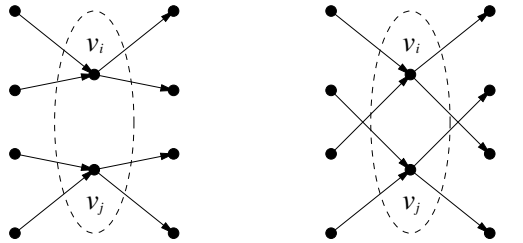
For a neighbor network χ' of a valid network χ, it is clear that $\widehat{|\chi'|} \le |\chi|$ implies $|R(E(\chi'))| \le |\chi|$. This means that, if the potential-length upper bound of χ' is not greater than the length of χ, the potential quality of χ' is not worse than the quality of χ. Therefore, we can replace χ with χ' and improve the network quality.

Furthermore, we approximate the quality of any network χ obtained in the process of local search by $\widehat{|\chi|}$. It is, of course, not true that $\widehat{|\chi|} \le \widehat{|\chi'|}$ always implies $|R(E(\chi))| \le |R(E(\chi'))|$. However, experiments showed that there is a strong correlation between $\widehat{|\chi|}$



(a) A sorting network χ (b) Its underlying graph $G(\chi)$

Figure 3: An example sorting network and its underlying graph



(a) A sorting network χ (b) A neighbor of χ

Figure 4: An illustration of exchanging process

and $|R(E(\chi))|$. When a tie occurs, we give favor to the networks with shorter edited lengths.

3.2.2 Underlying Graphs of Sorting Networks

Sorting networks have been generally represented in such a way as in Figure 1. Since it draws buses across comparators in order to focus on the data flows on buses, it is not appropriate for representing a physical situation involving discrete comparators and the relationship among them. We devised a more intuitive model to represent the relationship among comparators.

We set a vertex for each comparator. If an output of comparator c_i is fed into comparator c_j, we put an arc e_{ij} from the vertex v_i to the vertex v_j. We have a graph $G = (V, E)$ corresponding to a sorting network where V is the set of vertices and E is the set of directed edges. For a given network χ, we call such a graph an *underlying graph* and denote by $G(\chi)$. Figure 3 shows an example sorting network and its underlying graph.

3.2.3 Search Strategy

From the viewpoint of underlying graphs, the optimal sorting network problem can be considered to be the problem of finding the edges that optimally connects the vertices corresponding to given comparators — the problem of finding the optimal network topology. At first glance, it seems to be reasonable to do local search by creating and deleting arcs between two random vertices. However, for a network χ' obtained from a network χ in this manner, we have to consider all the binary sequences in the test set in order to get the value of $\widehat{|\chi'|}$ in most cases. This makes the local search intractable.

For this reason, we consider the networks, obtained by exchanging every two input bus indices of comparators in each layer of χ proceeding from left to right, as neighbor networks of χ. This exchanging process is equivalent to swapping two input-output pairs of the two vertices corresponding to two comparators in the same layer in the underlying graph $G(\chi)$. Figure 4 shows an example. Such a strategy considerably reduces the computational load of local search for the following reason.

Suppose that the number of parallel layers in χ is l, and let the parallel layers of χ be $\chi_0, \chi_1, \ldots, \chi_{l-1}$; let the functions corresponding to these parallel layers be $f_{\chi_0}, f_{\chi_1}, \ldots, f_{\chi_{l-1}}$. In general, if χ does not have any redundant comparator, the number of unsorted binary sequences steeply decreases as the parallel layers are added one by one. In other words,

$$|f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n)) - \mathcal{S}_n| \gg |f_{\chi_1}(f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n))) - \mathcal{S}_n| \gg \cdots \gg |f_{\chi_{l-1}}(f_{\chi_{l-2}} \cdots f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n)) \cdots) - \mathcal{S}_n| \,.$$

Therefore, in evaluating the qualities of neighbors, when we exchange two input bus indices for the comparators in the i^{th} layer of χ, we can considerably reduce the time of editing the neighbors and getting the numbers of n-pairs by considering only the sequences in $f_{\chi_{i-1}}(f_{\chi_{i-2}} \cdots f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n)) \cdots) - \mathcal{S}_n$ instead of all the sequences in $f_{\phi_{32}}(\mathcal{T}_n) - \mathcal{S}_n$. Such a strategy, which successively improves layers left to right, is natural in that a parallel layer strongly affects the subsequent layers [8].

In each layer, we improve the layer by exchanging input bus indices of comparators in the manner of sequential 2-opt [16] [19] [1]. Let n be the number of input bus indices and l the number of parallel layers of a network χ. And let χ_i be the i^{th} layer of χ $(0 \leq i \leq l-1)$. We denote by $exchange(\phi, a, b)$ the layer obtained by exchanging the a^{th} and b^{th} input bus indices of comparators in a parallel layer ϕ and by χ_{ϕ/χ_i} the network obtained by replacing a layer χ_i in χ with another layer ϕ. Figure 5 describes our local optimization algorithm.

4 GA Framework

We used a hybrid steady-state genetic algorithm. Figure 6 shows the outline of the hybrid genetic algorithm.

```
LocalOptimize(χ)
{
    for i ← 0 to l − 1 {
        Q ← ∅;
        χ_i^0 ← χ_i;
        for j ← 1 to n/2 {
            choose a, b ∈ {0, 1, ..., n − 1} − Q
                such that φ_j = exchange(χ_i^{j−1}, a, b)
                    and |χ_{φ_j/χ_i}| is maximal;
            Q ← Q ∪ {a, b};
            χ_i^j ← exchange(χ_i^{j−1}, a, b);
        }
        choose k ∈ {1, 2, ..., n/2}
            such that |χ_{χ_i^k/χ_i}| is maximal;
        χ ← χ_{χ_i^k/χ_i};
    }
    χ ← R(E(χ));
    return χ;
}
```

Figure 5: The outline of the local optimization

```
create initial population of a fixed size;
do {
    choose parent1 and parent2 from population;
    offspring ← crossover(parent1, parent2);
    mutation(offspring);
    edit(offspring);
    repair(offspring);
    local-optimization(offspring);
    replace(population, offspring);
} until (stopping condition);
return the best individual;
```

Figure 6: The outline of the hybrid genetic algorithm

The details are described in the following.

- **Encoding:** Each sorting network is represented by a chromosome. Figure 7 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 an integer corresponding to a comparator as in the function value table. Each parallel layer consists of a bounded number of independent comparators and the supplemental layer consists of an unlimited number of comparators.

  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; genes in the supplemental layer are appended by repair and local optimization. This is a variant of Baldwinian hybrid GAs [22] [14] [23].

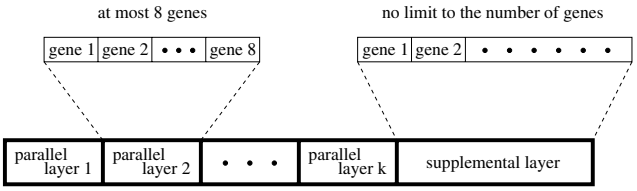


Figure 7: The structure of a chromosome in the 16-bus case

- **Initialization:** We set the population size to be 50. For each parallel layer in a chromosome, we randomly generate independent comparators. The number of independent comparators is chosen to be between a quarter and a half the number of input buses. We then perform the edit and repair processes to make the chromosome valid.

- **Parent 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),
  L_b : the length of the best (shortest), 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 layer independent of the other layers. Since the numbers of genes in the two parents are usually not the same, the traditional one-point crossover is not possible. Thus, we generate a cut point on the "relatively" similar position in the parents. Few parallel layers of offsprings made in this way are usually valid.[2] We convert each layer to a valid one by removing comparators until there is no comparator that shares the same bus with another comparator in the layer.

- **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, say c, in the same layer

[2]Here, a "valid" layer means a layer that consists of independent comparators. This usage is different from the other parts, e.g., Section 2.3, of this paper.

Table 1: Comparison of Experimental Results and Environments

	Hillis [13]	Drescher [9]	Juillé [15]†	Choi & Moon [8]	This Study
Population size	65,536	524,288	4,096	50	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, 100 % for 100 runs	60 comparators, 100 % for 100 runs
Execution time	5 to 50 min	5 to 18 min	5 to 10 min	2 to 15 min (average 5 min)	32 to 144 sec (average 72.5 sec)

†The version where the first 32 comparators are fixed

that occupies the changed bus, we connect the comparator c to the (necessarily) absent bus after change.

- **Edit, Repair and Local Optimization:** As mentioned in Section 2, we perform the edit and repair processes to an offspring after mutation. Then we perform the local search heuristic of Section 3 to it.
- **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 [6] and GENITOR-style replacement [24], and showed successful results in [5].

The genes in the parallel layers are updated as a result of local optimization while those in the supplemental layer are appended only for fitness evaluation. This is a combination of Lamarckian and Baldwinian GAs. This model can reduce the computational load of the Lamarckian side by ignoring some of the last comparators in the evolution.

5 Experimental Results

With the first 32-comparators of networks fixed, we evolved the population of networks using a genetic algorithm on an Intel Pentium III 866 MHz. We used a population size of 50, which is significantly smaller than other studies [13] [9] [15].

The chart in Figure 8 shows the performances of GAs according to the number of parallel layers used for the GAs. Each bar indicates the number of 60-comparator sorting networks (of quality equivalent to the best known) that the corresponding GA found in 144 seconds in 100 trials. When we used four parallels, the

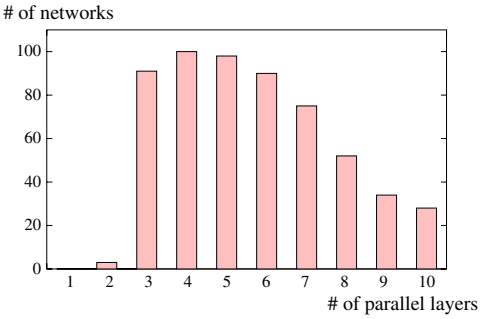


Figure 8: The number of 60-comparator sorting networks that we found in 144 seconds in 100 trials, according to the number of parallel layers

GA found 60-comparator networks in all of the 100 trials. On the other hand, when we used one parallel layer, we could not find any 60-comparator sorting network. The performance of the GA showed a bitonic distribution with respect to the number of parallel layers.

Table 1 summarizes the experimental results and the environments of Hillis [13], Drescher [9], Juillé [15], our previous study [8], and this study. This work showed the most stable performance in significantly less time even after considering the difference of CPU powers.

When we examined the 100 networks with 60 comparators that we found using 4 parallel layers, there were 72 distinct sorting networks.[3] Table 2 classifies the 72 sorting networks according to the number of parallel steps. We present in Figure 9 one of the 60-comparator sorting networks (with 10 parallel steps) that we found.

[3]We ignored the sequences in the same parallel step.

Table 2: The number of distinct sorting networks according to the number of parallel steps

# of parallel steps	10	11	12	Total
# of sorting networks	47	4	21	72

6 Conclusion

We paid attention to the relative importance of the comparison operations in the process of solving the sorting network problem. Experiments showed that these operations have a great effect on the performance. We designed a novel data structure in order to process the operations efficiently, and this led to the remarkable performance improvement.

To the best of our knowledge, no effective local search algorithm for the optimal sorting network problem has been proposed yet. We realized that this was because there was no efficient measure for evaluating the qualities of networks. We devised an approximate measure that not only reflects the qualities of networks sufficiently but also needs small computational time.

We also developed an effective local search heuristic based on this measure. In particular, we maximized the efficiency of the local search heuristic by reflecting the characteristics of the problem. Such an efficient and effective local search heuristic came to be another factor in the improvement of the performance.

A notable aspect of the proposed GA is that it evolves only a fixed number of parallel layers; the remaining layers are appended by the local optimization just for the evaluation of the solutions. This is a combination of a Lamarckian and a Baldwinian GA. It turned out that, when the chromosomal size of the Lamarckian side is properly determined, the Baldwinian side is easily optimized to a solution of best-known quality. This significantly reduces the problem search space and helps the GA to conduct an efficient search.

When the local optimization heuristic and the genetic algorithm were combined, they showed strong synergy. We found the best known solutions in 16-bus networks with a fairly small time budget. To the best of our knowledge, this is the most efficient result in the discovery of 60-comparator sorting networks.

We restricted the problem space by fixing the first 32 comparators. We are currently working on the theoretical justification of the restriction. The study includes experiments using the model without fixing any comparators. We will also consider greater-than-16 bus problems in the future.

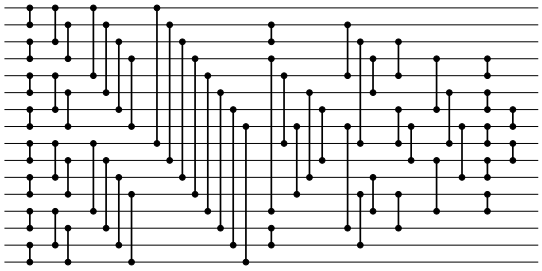


Figure 9: A 60-comparator sorting network that we found

Acknowledgements

This work was supported by KOSEF through the Statistical Research Center for Complex Systems at Seoul National University (SNU) and Brain Korea 21 Project. The RIACT at SNU provided research facilities for this study.

References

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

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

[3] 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.

[4] R. C. Bose and R. J. Nelson. A sorting problem. *Journal of ACM 9*, pages 282–296, 1962.

[5] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.

[6] D. Cavicchio. *Adaptive Search Using Simulated Evolution*. PhD thesis, University of Michigan, Ann Arbor, MI, 1970. Unpublished.

[7] S. S. Choi and B. R. Moon. A graph-based approach to the sorting network problem. In *Congress on Evolutionary Computation*, pages 457–464, 2001.

[8] S. S. Choi and B. R. Moon. A hybrid genetic search for the sorting network problem with evolving parallel layers. In *Genetic and Evolutionary Computation Conference*, pages 258–265, 2001.

[9] G. L. Drescher. Evolution of 16-number sorting networks revisited. Unpublished manuscript, 1994.

[10] 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.

[11] 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.

[12] M. W. Green. Some improvements in nonadaptive sorting algorithms. Technical report, Stanford Research Institute, Menlo Park, California, 1969.

[13] 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.

[14] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(3):495–502, 1987.

[15] 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.

[16] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

[17] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison Wesley, 1973.

[18] W. Li, J. Lin, and R. Unbehauen. On the analysis of sorting networks from the viewpoint of circuit theory. *IEEE Trans. on Circuits and Systems I — Fundamental Theory and Applications*, 45(5):591–593, 1998.

[19] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(4598):498–516, 1973.

[20] I. Pitas and A. N. Venetsanopoulos. A new filter structure for the implementation of certain classes of image processing operations. *IEEE Trans. on Circuits and Systems*, 35(6):636–647, 1988.

[21] G. Shapiro. In D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

[22] G. G. Simpson. The Baldwin effect. *Evolution*, 7:110–117, 1953.

[23] D. Whitley, V. Gordon, and K. Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In *International Conference on Evolutionary Computation*, pages 6–15, 1994.

[24] D. Whitley and J. Kauth. Genitor: A different genetic algorithm. In *Rocky Mountain Conf. on Artificial Intelligence*, pages 118–130, 1988.

Evolutionary Concept Learning

Federico Divina and Elena Marchiori
Computer Science Dept.
Vrije Universiteit, Amsterdam, The Netherlands
{divina,elena}@cs.vu.nl

Abstract

Inductive learning in First-Order Logic (FOL) is a hard task due to both the prohibitive size of the search space and the computational cost of evaluating hypotheses. This paper introduces an evolutionary algorithm for concept learning in (a fragment of) FOL. The algorithm evolves a population of Horn clauses by repeated selection, mutation and optimization of more fit clauses. Its main novelty, with respect to previous approaches, is the use of stochastic search biases for reducing the complexity of the search process and of the clause fitness evaluation. An experimental evaluation of the algorithm indicates its effectiveness in learning short hypotheses of satisfactory accuracy in a short amount of time.

1 Introduction

Learning from examples in FOL, also known as Inductive Logic Programming (ILP) (Muggleton & Raedt, 1994), constitutes a central topic in Machine Learning, with relevant applications to problems in complex domains like natural language and molecular computational biology (Muggleton, 1999).

Learning can be viewed as a search problem in the space of all possible hypotheses (Mitchell, 1982). Given a FOL description language used to express possible hypotheses, a background knowledge, a set of positive examples, and a set of negative examples, one has to find a hypothesis which covers all positive examples and none of the negative ones (cf. (Kubat et al., 1998; Mitchell, 1997)).

This problem is NP-hard even if the language to represent hypotheses is propositional logic. When FOL hypotheses are used, the complexity of searching is combined with the complexity of evaluating hypotheses (Giordana & Saitta, 2000).

Popular FOL learners like FOIL (Quinlan, 1990) and Progol (Muggleton, 1995) adopt a progressive coverage approach. One starts with a training set containing all positive and negative examples, construct a FOL (if-then) rule which covers some of the positive examples, removes the covered positive examples from the training set and continues with the search for the next rule. When the process terminates (after a maximum number of iterations or when all positive examples have been covered), the resulting set of rules is reviewed, e.g., to eliminate redundant rules. These algorithms use different greedy methods as well as heuristics (e.g. information gain) to cope with the complexity of the search.

FOL learners based on genetic algorithms act on more clauses at the same time. Systems like GIL (Janikow, 1993), GLPS (Leung & Wong, 1995) and STEPS (Kennedy & Giraud-Carrier, 1999) use an encoding where a chromosome represents a set of rules. In other GA based systems like SIA01 (Augier et al., 1995), REGAL (Giordana & Neri, 1996), G-NET (Anglano et al., 1998) and DOGMA (Hekanaho, 1998), a chromosome represents a clause. In the latter case a non redundant hypothesis is extracted from the final population at the end of the evolutionary process. Both approaches present advantages and drawbacks. Encoding a whole hypothesis in each chromosome allows an easier control of the genetic search but introduces a large redundancy, that can lead to populations hard to manage and to individuals of enormous size. Encoding one clause in each chromosome allows for cooperation and competition between different clauses hence reduces redundancy, but requires sophisticated strategies, like co-evolution, for coping with the presence in the population of super-individuals.

This paper introduces an evolutionary algorithm which evolves a set of chromosomes representing clauses, where at each iteration fitter chromosomes are selected, mutated, and optimized. The main novelty with respect to previous approaches is the introduction of two stochastic mechanisms for controlling the complexity of the construction, optimization and evaluation of clauses. The first mechanism allows the user to specify the percentage of background knowledge that the algorithm will use, in this way controlling the computational cost of fitness evaluation. The second mechanism allows one to control the greediness of the operators used to mutate and optimize a clause, thus controlling the computational cost of the search.

Furthermore we introduce and test a variant of the Universal Suffrage (US) selection operator ((Giordana & Neri, 1996)), called Weighted Universal Suffrage (WUS) selection operator. The US selection operator is based on the idea that individuals are candidates to be elected, and positive examples are the voters. Every positive example has the same voting power. The idea behind the WUS selection operator, is to give more voting power to examples that are harder to cover. The voting power of examples is adjusted during the computation.

We show experimentally that the algorithm is able to find hypotheses of satisfactory quality, both with respect to accuracy and simplicity, in a short time.

2 The Learning Algorithm

The algorithm considers Horn clauses of the form

$$p(X,Y) \leftarrow r(X,Z), q(Y,a).$$

consisting of atoms whose arguments are either variables (e.g. X, Y, Z) or constants (e.g. a). The atom $p(X,Y)$ is called head, and the set of other atoms is called body. The head describes the target concept, and the predicates of the body are in the background knowledge.

The background knowledge contains ground facts (i.e. clauses of the form $r(a,b) \leftarrow .$ with a, b constants). The training set contains facts which are true (positive examples) and false (negative examples) for the target predicate. A clause is said to *cover an example* if the theory formed by the clause and the background knowledge logically entails the example.

A clause has a declarative interpretation (universally quantified FOL implication)

$$\forall X, Y, Z(r(X,Z), q(Y,a) \rightarrow p(X,Y))$$

and a procedural one

in order to solve $p(X,Y)$ solve $r(X,Z)$ and $q(Y,a)$.

Thus a set of clauses forms a logic program, which can directly (in a slightly different syntax) be executed in the programming language Prolog. So the goal of the learning algorithm can be rephrased as finding a logic program that models a given target concept, given a set of training examples and a background knowledge.

The overall algorithm we introduce, called ECL (Evolutionary Concept Learner), is illustrated in pseudocode in the figure below.

```
ALGORITHM ECL
Sel = positive_examples
repeat
    Select partial Background Knowledge
    Population = Initial_pop
    while (not terminate) do
        Select n chromosomes using Sel
        for each selected chromosome chrm
            Mutate chrm
            Optimize chrm
            Insert chrm in Population
        end for
    end while
    Store Population in Final_Population
    Sel = Sel - { positive examples
        covered by clauses in Population }
until max_iter is reached
Extract final theory from Population
```

In the repeat statement the algorithm constructs iteratively a `Final_population` as the union of max_iter populations. At each iteration, part of the background knowledge is chosen using a stochastic search bias described below.

A `Population` is evolved by the repeated application of selection, mutation and optimization (the `while` statement). These operators use only the chosen part of background knowledge.

At each generation of the evolution, n clauses are selected by means of the Universal Suffrage (US) selection operator (Neri & Saitta, 1995), a powerful selection mechanism used for achieving species formation in GAs for concept learning. US selection chooses randomly a positive example from the set `Sel` of positive examples yet not covered by clauses in the actual `Final_population` , and performs a roulette wheel selection on those clauses of the `Population` which cover that example. If the example is not yet covered by

any clause, a new clause is constructed for that example using a seeding operator. The selected clause is then modified using the mutation and optimization operators, and is inserted in the population.

When the construction of the `Final_population` is completed, a logic program is extracted using a set covering algorithm.

Before presenting the main steps of ECL, we describe the stochastic search biases.

2.1 Stochastic Search Biases

ECL uses two stochastic mechanisms, one for selecting part of the background knowledge, and one for selecting the degree of greediness of the operators used in the evolutionary process.

A parameter p (p real number in $(0, 1]$) is used in a simple stochastic sampling mechanism which selects an element of the background knowledge with probability p. In this way the user can limit the cost of the search and fitness evaluation by setting p to a low value. This because only a part of the background knowledge will be used when assessing the goodness of an individual. This leads to the implicit selection of a subset of the examples (only those examples that can be covered with the partial background knowledge selected will be considered). Individuals will be evaluated on these examples using only the partial background knowledge. In this way an individual can be wrongly evaluated because a subset of the examples is used, and also because those examples can be wrongly classified, in case they are covered using the whole background knowledge, but are not covered using the partial background knowledge. This is different from other mechanisms used for improving the efficiency of fitness evaluation, like (Glover & Sharpe, 1999), (Teller & Andre, 1997), where training set sampling is employed for speeding up the evaluation of individuals.

The construction, mutation and optimization of a clause use four greedy generalization/specialization operators (described later in an apart section). Each greedy operator involves the selection of a set of constants (or of a set of variables). The size of this set can be supplied by the user by setting a corresponding parameter Ni ($i = 1, \ldots, 4$). The elements of the set are then randomly chosen with uniform probability. In this way the user can control the greediness of the operators, where higher values of the parameters imply higher greediness.

Finally ECL uses also a language bias which is commonly employed in ILP systems for limiting explicitly the maximum length of clauses.

These search biases allow one to reduce the cost of both the search and fitness evaluation, but the price to pay may be the impossibility of finding the best clauses.

2.2 Fitness and Representation

The quality of a clause cl is measured by the following fitness function:

$$fitness(cl) = \frac{pos - pos_{cl}}{pos} + w \cdot \frac{neg_{cl}}{neg}$$

The aim of ECL is to evolve clauses with minimum fitness, that is which cover many positive examples and few negative ones. In the above formula pos and neg are respectively the total number of positive and negative training examples, pos_{cl}, neg_{cl} are the number of positive and negative examples covered by the clause cl, and w is a weight used to favor clauses covering few negative examples. The weight w is used to deal with skewed distributions of the examples, where a high weight is used when there are much more positive examples than negative ones.

ECL uses a high level representation similar to the one used by SIA01 (Augier et al., 1995), where a clause $p(X, Y) \leftarrow r(X, Z), q(Y, a).$ is described by the sequence

$$\boxed{p, X, Y}\ ,\ \boxed{r, X, Z}\ ,\boxed{q, Y, a}$$

This representation is preferred to other GA typical representations, like bit string, because it allows the direct use of ILP operators for generalization and specialization of a clause. Moreover, it does not constraint the length of a chromosome, like e.g in the bitwise representation used in the REGAL and G-NET systems, which requires the user to specify an initial template for the target predicate.

2.3 Clause Construction

A clause cl is constructed when the US selection operator selects a positive example which is not yet covered by any clause in the actual population. This example is used as seed in the following procedure, where BK_p denotes the chosen part of background knowledge.

1. The selected example becomes the head of the emerging clause;
2. Construct two sets A_{cl} and B_{cl}. A_{cl} consists of all atoms in BK_p having *at most* one argument which does *not* occur in the head; B_{cl} contains all elements in $BK_p \setminus A_{cl}$ having at least one argument occurring in the head.

3. **while** $length(cl) < l$ and $A_{cl} \cup B_{cl} \neq \emptyset$
   (a) Randomly select an atom from A_{cl} and remove it from A_{cl}. If A_{cl} is empty then randomly select an atom from B_{cl} (and remove it from B_{cl}). Add the selected atom to the emerging clause cl.
4. Generalize cl as much as possible by means of the repeated application of the generalization operator 'constant into variable' (described in the next section). Apply this operator to the clause until either its fitness increases or a maximum number of iterations is reached. In the former case, retract the last application of the generalization operator.

In step 3 l is the maximum length of a clause, supplied by the user. If l was not supplied then the first condition of the *while* cycle is dropped, and no constraint on the length of the clause is imposed.

2.4 Selection

The US selection operator, first introduced in (Giordana & Neri, 1996), selects clauses in two steps:

1. randomly select n examples from the positive examples set;
2. for each selected example e_i, $1 \leq i \leq n$, let $Cov(e_i)$ be the set of clauses in the current population that cover e_i. If $Cov(e_i) \neq \emptyset$, choose one clause from $Cov(e_i)$ using a roulette wheel mechanism, where the sector associated with the clause $c \in Cov(e_i)$ is proportional to the ratio between the fitness of c and the sum of the fitness of all the clauses occurring in $Cov(e_i)$. If $Cov(e_i) = \emptyset$ create a new clause covering e_i, using e_i as a seed.

When introduced, in (Giordana & Neri, 1996), the US selection operator was used in a distributed system, made of various genetic nodes, where each genetic node performs a GA. In that setting, the examples assigned to each node were different, and the training sets changed during the computation. However at the GA level the examples were the same, and had the same probability of being selected.

We propose here the following variant of the US selection, called *Weighted US selection*, where examples have different probability of selection. A weight is associated to each example, where smaller weights are associated to examples harder to cover. Then the random selection used in step 1 of the US selection above is replaced by a selection which takes into account the weights of examples.

More in detail, the weight of an example e is equal to

$$\frac{\mid Cov(e) \mid}{\mid Pop \mid}$$

being Pop the current population and $Cov(e)$ the set of clauses of Pop that cover e. If the population is empty, then every example has the same weight.

The examples are then selected with a roulette wheel mechanism, where the dimension of the sector associated to each examples is inversely proportional to the weight of the example. So the less clauses cover an example, the more chances that example has of being selected. The weights of the examples are updated at every iteration. Once the examples have been selected, the selection of the clauses is made as in the standard US selection operator.

With this mechanism not only uncovered examples are favored, but also examples that are covered by few clauses are favored, having wider sector in the roulette wheel. Examples covered by many clauses are penalized, because easier to cover. In this way the system will focus at each iteration more and more on the harder examples to be covered.

2.5 Mutation and Optimization

The mutation consists of the application of one of the four generalization/specialization operators. This operator is chosen as follows. First, a (randomized) test decides whether it will be a generalization or a specialization operator. Next, one of the two operators of the chosen class is randomly applied. The first test is based on the completeness and the consistency of the selected individual. If the individual is consistent with the training set, then it is likely that the individual will be generalized. Otherwise it is more probable that the individual will be specialized. The test decides to generalize a clause cl with probability

$$p_{gen}(cl) = \frac{1}{2}(\frac{pos_{cl} - negcl}{pos + neg} + \alpha)$$

otherwise it decides to specialize the clause. The constant α is used to slightly bias the decision toward generalization. The probability p_{gen} is maximal when cl covers all positive and no negative examples, and it is minimal in the opposite case.

The optimization phase consists of a repeated application of the greedy operators to the selected individual, until either its fitness does not increase or a maximum number of iterations is reached.

The system does not make use of any crossover operator. Experiments with a simple crossover operator,

which uniformly swaps atoms of the body of the two clauses, have been conducted. However the results did not justify its use.

2.6 Hypothesis Extraction

The termination condition of the main **while** statement of ECL is met when either all positive examples are covered or a maximum number of iterations is reached. In this case a logic program for the target predicate is extracted from the final population. The theory has to cover as many positive examples as possible, and as few negative ones (notice that at this stage the clauses have been "globally" evaluated, using the complete background knowledge). This problem can be translated into an instance of the weighted set covering problem as follow. Each element cl of the final population is a column with positive weight equal to

$$weight_{cl} = neg_{cl} + 1$$

and each covered positive example is a row. The columns relative to each positive example are the clauses that cover that example. In this way clauses covering few negative examples are favored. A fast heuristic algorithm ((Marchiori & Steenbeek, 2000)) is applied to this problem instance to find a "best" theory.

3 Generalization/Specialization Operators

A clause cl is generalized either by deleting an atom from the body of the clause or by replacing (all occurrences of) a constant with a variable. Dually, cl is specialized by either adding an atom to the body of cl, or by replacing (all occurrences of) a variable with a constant.

The four operators utilize four parameters $N_1, \ldots, N_4$, respectively, in their definition, and a gain function. When applied to operator τ and clause cl, the gain function yields the difference between the clause fitness before and after the application of τ:

$$gain(cl, \tau) = fitness(cl) - fitness(\tau(cl))$$

The four operators are defined below.

3.1 Atom Deletion

Consider the set Atm of N_1 atoms of the body of cl randomly chosen. For each A in Atm, compute $gain(cl, -A)$, the gain of cl when A is deleted from cl.

Choose an atom A yielding the highest gain $gain(cl, -A)$ (ties are randomly broken), and generalize cl by deleting A from its body.

Insert the deleted atom A in a list D_{cl} associated with cl containing atoms which have been deleted from cl. Atoms from this list may be added to the clause during the evolutionary process by means of a specialization operator.

3.2 Constant into Variable

Consider the set Var of variables present in cl plus a new variable. Consider also the set Con consisting of N_2 constants of cl randomly chosen.

For each a in Con and each X in Var, compute $gain(cl, \{a/X\})$, the gain of cl when all occurrences of a are replaced by X.

Choose a substitution $\{a/X\}$ yielding the highest gain (ties are randomly broken), and generalize cl by applying $\{a/X\}$.

3.3 Atom Addition

Consider the set Atm consisting of N_3 atoms of B_{cl} (list built at initialization time) and of N_3 atoms of D_{cl}, all randomly chosen.

For each A in Atm compute $gain(cl, +A)$, the gain of cl when A is added to the body of cl.

Choose an atom A yielding the highest gain $gain(cl, +A)$ (ties are randomly broken), and specialize cl by adding A to its body.

Remove A from its original list (B_{cl} or D_{cl}).

3.4 Variable into Constant

Consider the set Con consisting of N_4 constants (of the problem language) randomly chosen, and a variable X of cl randomly chosen.

For each a in Con, compute $gain(cl, \{X/a\})$, the gain of cl when all occurrences of X are replaced by a.

Choose a substitution $\{X/a\}$ yielding the highest gain (ties are randomly broken), and specialize cl by replacing all occurrences of X with a.

4 Experimental Evaluation

We consider three datasets for experimenting with ECL: the vote, credit and mutagenesis dataset, respectively. The three dataset are public domain datasets.

The vote dataset contains votes for each of the U.S. House of Representatives Congressmen on the sixteen key votes. The problem is learning a concept for distinguishing between democratic and republican congressmen. The dataset consists in 435 instances, of which 267 are examples of democrats, and 168 are republicans.

The credit dataset concerns credit card applications. The problem consists in learning when to allow a subject to have a credit card or not. There are 690 instances, of which 307 are positive instances and 383 are negative instances. Each instance is described by fifteen attributes. These first two datasets are taken from (Blake & Merz, 1998).

The mutagenesis dataset comes from the field of organic chemistry, and concerns the problem of learning the mutagenic activity of nitroaromatic compounds. These compounds occur in automobile exhaust fumes and are also common intermediates in the synthesis of many thousands of industrial compounds (Debnath et al., 1991). Highly mutagenic nitroaromatics have been found to be carcinogenic (Ashby & Tennant, 1991). The concept to learn is expressed by the predicate *active(C)*, which states that compound C has mutagenic activity. The dataset originates from (Debnath et al., 1991).

The parameter settings used in the experiments are given in Table 1.

	Vote	Credit	Mutagenesis
pop_size	80	20	50
mut_rate	1	1	1
n	10	2	7
max_gen	5	30	10
max_iter	2	10	10
N(1,2,3,4)	(3,6,2,5)	(2,5,2,5)	(4,8,2,8)
p	0.1	0.2	0.1
l	4	4	8

Table 1: Parameter settings: pop_size = maximum size of population, mut_rate = mutation rate, n = number of selected clauses, max_gen = maximum number of GA generations, max_iter = maximum number of iterations, N(1,2,3,4) = parameters of the four greedy operators, p= parameter of BK selection, l = maximum length of a clause

These values have been obtained after a few experiments on the training sets, with the constraint that a run of ECL would take at most 1 hour. As expected, the values found depend on the specific dataset. Unfortunately, we were unable to find general rules that could explain the choice of these parameters. This is in general a challenging problem (Eiben et al., 1999), and we are actually investigating methods for the on-line adaptation of parameters.

The evaluation method used is ten-fold cross validation. Each data set is divided in ten disjoint sets of similar size; one of these sets is used as test set, and the union of the remaining nine forms the training set. Then ECL is run on the training set and it outputs a logic program, whose performance on new examples is assessed using the test set.

This process is repeated 10 times, using each time a different set as test set. The average of all the results is taken as final evaluation measure for ECL.

We consider three performance measures:

- efficiency: the running time of the algorithm on the training set for finding the logic program;
- simplicity: the number of clauses of the logic program;
- accuracy: the proportion of examples in the test set which have been correctly classified by the resulting logic program.

System	Vote	Credit	Mutagenesis
G-NET	0.95 (0.032)	0.84 (0.044)	0.91 (0.079)
C4.5	0.95 (0.030)	0.86 (0.033)	n.a.
Progol	-	-	0.80 (0.030)
ECL	0.94 (0.023)	0.79 (0.072)	0.87 (0.056)

Table 2: Accuracy results obtained using ten-fold cross validation. Standard deviation is given between brackets.

Results obtained by ECL are compared to results obtained by three of the most effective concept learners based on different approaches in table 2. C4.5 is based on decision trees, Progol employs a progressive coverage method, and G-NET is a distributed coevolutionary genetic algorithm.

The results for the first three systems are taken from (Anglano et al., 1998), while the result of Progol is taken from (Srinivasan et al., 1994). All the results were obtained using the same ten-fold cross validation. On the vote dataset the results obtained by ECL are comparable to those obtained with the other three systems. The results on the credit dataset are worse than those of G-NET and C4.5.

Table 3 presents the results obtained on the three datasets using the parameter p set to one. This means

Dataset	Accuracy	Efficiency	Simplicity
Vote	0.94 (0.033)	66 min	5.89
Credit	0.71 (0.074)	224min	41.1
Mutagenesis	0.81 (0.089)	81 min	16

Table 3: Results obtained by ECL using the same parameters shown in table 1, but here p is set to 1, so that the whole background knowledge is used.

Dataset	Efficiency	Simplicity
Vote (ECL)	29 minutes	5
Vote (G-NET)	-	2
Credit (ECL)	50 minutes	5
Credit (G-NET)	-	14
Mutagenesis (ECL)	10 minutes	4
Mutagenesis (GNET)	few hours	3

Table 4: Efficiency = average running time, Simplicity= average number of clauses for ECL, and of disjuncts for G-NET.

that the whole background knowledge will be used. The other parameters are the ones defined in table 1. It can be seen that the results are not better than the results shown in table 2, especially in the mutagenesis dataset. This is probably due to overfitting that can take place when too much information about the problem to tackle is present. Moreover, as expected, the system slows down sensibly when using the whole background knowledge.

Table 4 shows the average time taken by a run on each dataset. Table 4 shows that even if in some cases other systems outperform ECL, ECL is able to find, in a short amount of time, a simple result with a satisfactory accuracy. For instance, on the mutagenesis dataset ECL is able to find a simple logic program with 4 clauses in just 10 minutes (on the average). In contrast, as mentioned in (Anglano et al., 1998), G-NET, which is a distributed system working on a cluster of workstations, needs few hours for finding a theory of comparable simplicity, which is not much more accurate. Unfortunately detailed results on the execution time of G-NET were not available, and also replicating the experiments resulted not possible, due to the impossibility to install the system[1].

4.1 US vs. Weighted US Selection

Table 5 reports some results in which the US and the WUS selection operators are compared. It can been

[1]Thanks to F. Neri for his support.

Operator	WUS	US
Vote	0.941	0.882
Credit	0.790	0.795
Mutagenesis	0.872	0.860

Table 5: Average accuracy results obtained on the three dataset, with US and weighted US selection operator.

seen that the use of the WUS selection operator improves the accuracy of the system for the vote and the mutagenesis datasets. In particular in the vote dataset the difference is evident. For the credit dataset the use of the WUS selection operator does not lead to any improvement. Even if the results of the experiments do not indicate a dramatic benefit of the WUS selection operator over the US one, it does not affect the efficiency of the system hence it can be used as alternative selection mechanism.

5 Conclusion

In this paper we presented a concept learning algorithm based on evolutionary computation, which incorporates novel simple parametric mechanisms for controlling the cost of searching the hypotheses space and the cost of fitness evaluation. We also introduced a variant of the US selection operator, called Weighted US selection operator.

The algorithm can be used profitably for exploring efficiently a new learning problem to get a first rough idea of possible simple models of the target concept, or for experimenting with a range of different search strategies at the same time, including random search and hill climbing as bounds of the range, which can be obtained from ECL by setting appropriately the bias search parameters.

The search biases of ECL assume a uniform distribution of the data used for selection. This does not reflect reality in many learning problems. We are actually investigating alternative stochastic sampling mechanisms for selecting the portion of background knowledge, which would take into account the estimated importance of each element (e.g. fact of the background knowledge) according to some evaluation measure obtained from the examples in the training set.

References

Anglano, C., Giordana, A., Bello, G. L., & nza Saitta, L. (1998). An experimental evaluation of coevolutive

concept learning. *Proc. 15th International Conf. on Machine Learning* (pp. 19–27). Morgan Kaufmann, San Francisco, CA.

Ashby, J., & Tennant, R. (1991). Definitive Relationships among chemical structure, carcinogenicity and mutagenicity for 301 chemicals tested by the U.S. NTP . *Mutation Research*, *257*, 229–306.

Augier, S., Venturini, G., & Kodratoff, Y. (1995). Learning first order logic rules with a genetic algorithm. *The First International Conference on Knowledge Discovery and Data Mining* (pp. 21–26). Montreal, Canada: AAAI Press.

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Debnath, A., de Compadre, R. L., Debnath, G., Schusterman, A., & Hansch, C. (1991). Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medical Chemistry*, *34(2)*, 786–797.

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in Evolutionary Algorithms. *IEEE-EC*, *3*, 124.

Giordana, A., & Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation*, *3*, 375–416.

Giordana, A., & Saitta, L. (2000). Phase Transitions in Relational Learning. *Machine Learning*, *41(2)*, 217–251.

Glover, R., & Sharpe, P. (1999). Efficient GA based techniques classification. *Applied Intelligence*, *11(3)*, 277–289.

Hekanaho, J. (1998). DOGMA: a GA based relational learner. *Proceedings of the 8th International Conference on Inductive Logic Programming* (pp. 205–214). Springer Verlag.

Janikow, C. (1993). A knowledge intensive genetic algorithm for supervised learning. *Machine Learning*, *13*, 198–228.

Kennedy, C. J., & Giraud-Carrier, C. (1999). A depth controlling strategy for strongly typed evolutionary programming. *GECCO 1999: Proceedings of the First Annual Conference* (pp. 1–6). Morgan Kauffman.

Kubat, M., Bratko, I., & Michalski, R. (1998). A review of Machine Learning Methods. In R. Michalski, I. Bratko and M. Kubat (Eds.), *Machine learning and data mining*. Chichester: John Wiley and Sons Ltd.

Leung, K., & Wong, M. (1995). Genetic logic programming and applications. *IEEE Expert*, *10(5)*, 68–76.

Marchiori, E., & Steenbeek, A. (2000). An Evolutionary Algorithm for Large Scale Set Covering Problems with Application to Airline Crew Scheduling. *Real World Applications of Evolutionary Computing. Springer* (pp. 367–381). Springer-Verlag.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203–226.

Mitchell, T. (1997). *Machine learning*. Series in Computer Science. McGraw-Hill.

Muggleton, S. (1995). Inverse entailment and PROGOL. *New Generation Computing*, *13*, 245–286.

Muggleton, S. (1999). Inductive logic programming: issues, results and the challenge of learning language in logic. *Artificial Intelligence*, *114*, 283–296.

Muggleton, S., & Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19-20*, 669–679.

Neri, F., & Saitta, L. (1995). Analysis of genetic algorithms evolution under pure selection. *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 32–39). Morgan Kaufmann, San Francisco, CA.

Quinlan, J. (1990). Learning logical definition from relations. *Machine Learning*, *5*, 239–266.

Srinivasan, A., Muggleton, S., King, R., & Sternberg, M. (1994). Mutagenesis: Ilp experiments in a nondeterminate biological domain. *Proceedings of the 4th International Workshop on Inductive Logic Programming* (pp. 217–232). Gesellschaft für Mathematik und Datenverarbeitung MBH.

Teller, A., & Andre, D. (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. *Genetic Programming 1997: Proceedings of the Second Annual Conference* (pp. 321–328). Stanford University, CA, USA: Morgan Kaufmann.

The Effect of Cost Distributions on Evolutionary Optimization Algorithms

César Galindo-Legaria
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052
USA
cesarg@microsoft.com

Florian Waas⋆
CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands
flw@cwi.nl

Abstract

According to the No-Free-Lunch theorems of Wolpert and Macready, we cannot expect one generic optimization technique to outperform others on average [WM97]. For every optimization technique there exist "easy" and "hard" problems. However, little is known as to what criteria determine the success of an optimization technique.

In this paper, we consider this question from the evolutionary computing point of view. We use cost distributions, i.e., the frequencies of the objective function's values occurring in the search spaces, to devise a classification of optimization problems. Unlike fitness landscapes, the cost distribution is truly problem intrinsic rathern than part of an algorithmic solution.

Based on the characteristic cost distribution of a problem, our model helps to predict what components of an evolutionary algorithm are most relevant (e.g., initialization, mutation), and what is the expected overall performance. We validate the model through experiments on three problems: Set Partitioning, Knapsack, and Traveling Salesman.

1 INTRODUCTION

Evolutionary algorithms have been shown to be very successful for a variety of optimization problems, for example timetabling and scheduling. However, there are a number of other problems, like Traveling Salesman, that have been much more difficult to tackle using the same techniques.

There is some intrinsic property of a problem that makes it more or less suitable to be tackled by evolutionary algorithms. In the past, researchers have observed three major classes of NP-complete optimization problems:

Easy. Near-optimal or even optimal solutions can be found without major difficulties. In this case, sophisticated mutation or crossover operations have only marginal impact on the quality of the result. In a sense, these problems are "too easy" for the evolutionary framework. Simpler optimization algorithms like Hill Climbing often outperform evoluationary techniques in that they find results of comparable quality much quicker. An example of this kind of problem is Set Partitioning.

Adequate. A large class of problems belong to this class where evolutionary algorithms excel, often outperforming other optimization strategies significantly. One example of this kind of problem is Knapsack.

Hard. Problems like the Traveling Salesman Problem pose particular difficulties to evoluationary optimization. In this case sophisticated, problem-specific tuning is necessary to obtain acceptable results. In contrast to *easy* problems, evolutionary optimization seldom finds optimal solutions. Most interestingly, this particular difficulty appears to be of a general nature, independent of the type of evolutionary optimization used or topology defined in the search space.

In this paper we investigate the role of *cost distributions* of optimization problems. By this we mean the frequencies of all occurring values of the objective function throughout the entire search space. It provides basic statistical information on average cost of a random solution, and concentration of good or bad solutions. Cost distributions are independent of any notion of adjacency, proximity, or neighborhood of solutions, defined for example as an algorithmic transformation between solutions. Rather, given a problem instance,

⋆Author's current address: Microsoft Corp, Redmond, WA 98052, USA

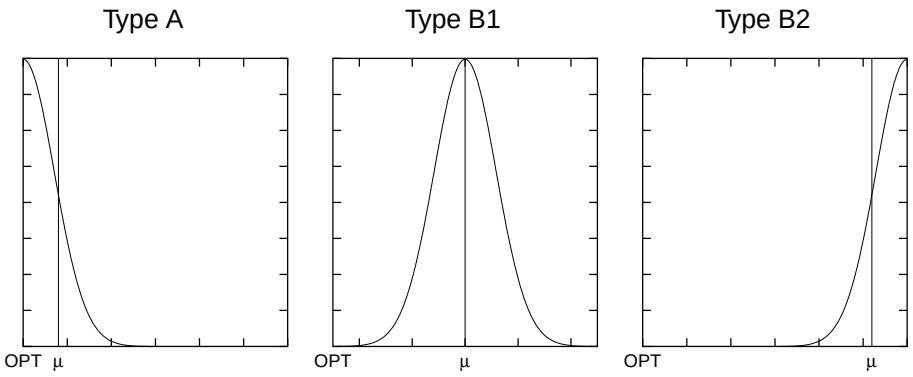


Figure 1: *Basic types of distributions (qualitatively). Optimum denoted by OPT, mean by μ;*

its cost distribution underlies *all* possible topologies a search algorithm can define on the search space.

Through a large number of experiments, we have seen that cost distributions appear to be very characteristic for different optimization problems [Waa99, WGL00a]: Different instances of a given problem exhibit cost distributions which are of similar quality. The variation within a problem is gradual and limited.

We present here a broad classification of cost distributions, based on surveying the literature and own experimental observations. We discuss how each building block of the evolutionary framework (e.g., initialization, mutation) is affected by the cost distribution, and what is the overall impact on the optimization algorithm. Our model helps explain earlier results on the behavior of evolutionary optimization. Also, we believe our analysis of individual components can aid in tailoring the general evolutionary framework to specific problems, and our general model will be useful to predict how amenable are new problems for treatment using evolutionary techniques.

2 PARAMETERS OF SEARCH SPACE

Since the introduction of general search algorithms, significant effort has been devoted to characterize the *search space*—i.e., the set of all possible solutions—and its influence on the search algorithms. In this section we discuss cost distributions and how they influence topological models.

2.1 COST DISTRIBUTIONS

A cost distribution captures the frequencies of cost values—i.e., values of the objective function—in the complete search space. [1] For any particular cost c, the distribution indicates the number of feasible solutions in the space whose cost is c. This information is the basis to answer questions such as: Are there "many solutions" close to the optimum?

When a space of solutions is too large to be enumerated, the general shape of the cost distribution can be approximated by uniform random sampling of the space (or quasi-random sampling like random walks, when uniform sampling is too hard).

Cost distributions are independent of the algorithm used to tackle the problem and are an invariant property of the particular problem instance. No matter whether a topology is defined at a later stage, the cost values and their frequencies are not altered.

What makes cost distributions such an important instrument is the possibility to analyze concentrations of cost values in the search space. We are in particular interested in the distance between the optimum and the bulk of solutions. Without loss of generality we assume a minimization problem. The question central to our further considerations is therefore:

> *Is the bulk of solutions close to the optimum or is the optimum an outlier with respect to the distribution?*

In large test series with different optimization problems, we have observed two basic types A and B of cost distributions, the second of which comes as two sub-types B1 and B2. In Figure 1, these distributions are shown qualitatively.

In problems with type-A distribution, the bulk of solu-

[1] We prefer the term *cost* over *fitness*, because sometimes fitness is intended as a relative measure (see e.g., [ZT99]). Instead, cost refers to the absolute value of the objective function.

tions is very close to the optimal costs, i.e., there are many optimal or near-optimal solutions in the search space. The optimum can even have highest frequency of all solutions (see Sec. 4.1). Note that this cost-wise proximity does not imply any neighborhood or topology, but simply indicates that many different solutions with similar cost exist.

In problems with type-B distribution, the bulk of solutions is of distinctly different cost than the optimum. We can distinguish the sub-types B1 where the mean is at a moderate distance of the optimum, and B2 where the bulk of solutions is far away from the optimum, i.e., the optimum is an outlier and near-optimal solutions are rare.

Note, the actual shape of the distribution–symmetry, skew, etc.–is of little relevance. The concentration of solutions relative to the optimum will be important for the further analysis. In practical problems cost distributions will likely show disturbances, and certainly not all problems can be assigned exactly to one type but may be in between two types. Yet, we can identify clear trends and the results of our analysis can be interpolated as necessary.

2.2 WHAT FITNESS LANDSCAPE?

Models for the *topology* or *landscape* of the space are powerful tools to interpret certain effects occurring with optimization algorithms (see e.g., [Kau93]). However, unlike the cost distribution, the space landscape is not intrinsic to the problem, and completely different topologies can be defined for a given space.

For instance, consider the Traveling Salesman Problem, where the shortest tour via a number of cities is sought. Let us define two different notions of neighborhood N1 and N2. Two tours are neighbored if one can be transformed into the other by

N1: exchanging two subsequently visited cities;

N2: exchanging *any* two cities;

Figure 2 illustrates the consequences with the *optimal* tour and possible neighbors according to the two different neighborhood relations. Whereas neighbors under N1 are of very similar cost, neighbors under N2 can be of higher differences in costs. Moreover, N2 is a super set of N1, i.e., neighbors in N1 are also neighbors in N2 but not conversely.

Both N1 and N2 induce a topology, thus a landscape, on the search space. One appears relatively smooth (N1), the other rugged (N2). Yet they are both defined on the same problem. Neither of the two landscapes is intrinsic to the problem, and neither could be claimed to be "the natural" landscape. A number of other neighborhood relations for this problem have been described in literature; for a survey on neighborhood defining transformations see for instance [FJMO95].

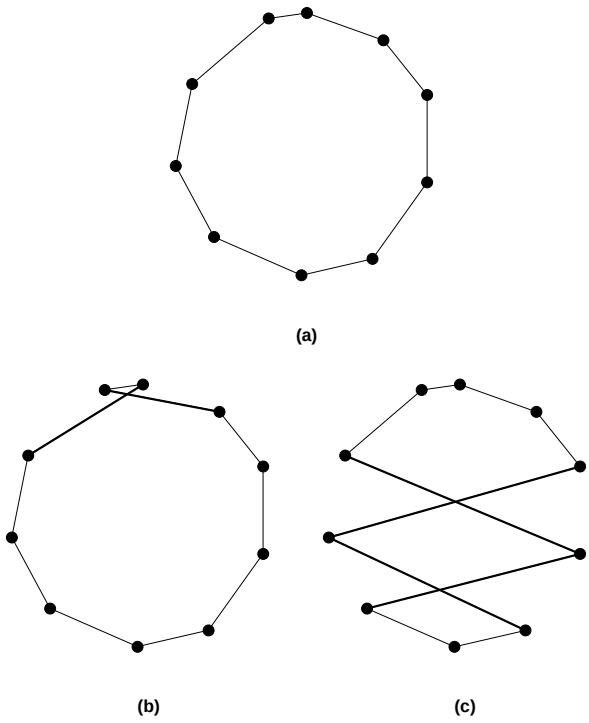


Figure 2: *Alternative tours for a Traveling Salesman Problem; (a) optimal tour, (b) tour neighbored to optimal tour under N1, (b) under N2*

In our view, the space landscape is part of the approach to solve a problem. In contrast, the cost distribution is an intrinsic attribute of the problem. A main claim of this paper is that useful insight can be gained from considering the cost distribution alone, and that such insight holds regardless of the specific topology imposed on the space.

3 PRINCIPLES OF EVOLUTIONARY ALGORITHMS

The notion of evolutionary computing is fairly flexible, comprising a large variety of algorithms and techniques. Frameworks as for instance presented in [Gol89, Mit96] are capable of simulating other algorithms that are commonly not considered evolutionary, like Random Sampling or Simulated Annealing [Bäc96]. On the other hand there are several generic elements that are agreed to be characteristic for an evo-

	Type-A	Type-B1	Type-B2
Initialization	$\oplus$	$\odot$	$\odot$
Recombination	$\oplus$	$\oplus$	$\oplus\oplus$
Mutation	$\oplus$	$\odot$	$\ominus$
Restarts	$\oplus$	$\odot\ominus$	$\odot\ominus$
Overall	easy	adequate	hard

$(\oplus)\oplus$ = *(strong) positive influence,* $\odot$ = *no influence,* $\ominus$ = *negative influence*

Table 1: *Importance of components of evolutionary algorithms with respect to the cost distribution*

lutionary algorithm. In our analysis, we first sketch this generic key elements and scrutinize the impact of cost distributions on those components. In particular, we investigate to what degree the single components use randomly selected solutions. Random sampling, uniform or biased, on—possibly restricted—sets of solutions is the very nucleus of all randomized optimization algorithms including evolutionary techniques.

Starting with a randomly generated initial population, generations are repeatedly derived by selecting a set of parents, generating the offspring by *recombination*, introducing a certain random distortion in form of *mutation*, and subjecting all individuals to a *selection* process. The algorithm terminates as soon as a certain stopping criterion—e.g., timeout, maximum number of individuals reached, or no improvement over a certain number of generations—is fulfilled. In every generation, all individuals are checked for their *fitness*, i.e., their costs, not only for the selection of the next generation but also to keep track of the best individual found so far. Simulating the natural evolutionary process the algorithm achieves a gradual improvement concentrating on well suited individuals by selection and the production of closely related offspring.

Initialization. The influence of the cost distribution on the initial phase is significant as *initializing* directly translates to *sampling*. Note, that sampling here does not necessarily mean uniform sampling.

For a type-A distribution the probability to find already near-optimal solutions in the initial sample is high. In other words, the subsequent optimization phase cannot improve the initially found solutions substantially. The probability that high quality solutions are included in the initial solution depend further on the size of the population: *very* small populations may differ enormously in quality.

In case of a type-B distribution, the initialization's role is less important, depending on the distance of the cost of the optimal solution from the average cost. The sampled initial individuals are of comparable, distinctly sub-optimal quality. In type-B2 problems, the initialization produces only results of constant but low quality. As opposed to the previous case, the size of the population does not affect its quality—the probability to sample a near-optimal solution is virtually zero.

Recombination. Implementing a mating between two individuals results in a *random* solution which consists of parts of its ancestor.

In the case of the type-A distribution sophistication is usually of limited use only as there are plenty of solutions in the close vicinity. However, if there are too many close relatives, guiding the recombination process becomes also more difficult.

The less solutions with similar costs to their ancestors there are, the more astray—i.e., in direction of the average cost—the recombination may lead. More sophisticated algorithms are necessary to avoid a fall back to the bulk of solutions in case of a type-B2 distribution.

Mutation. In case of a type-A distribution, mutation can be most fruitful as the odds to improve by random alteration are high.

For a type-B distribution, the probability to achieve an immediate improvement by mutation is very small but mutation is still useful to avoid undue concentration of certain properties among the individuals.

Restarts. Evolutionary algorithms, mimicking the natural evolutionary process are characterized by convergence, i.e., the overall fitness of the consecutive generations increases—although it is not necessarily monotonic. For simplified models of those algorithms, the convergence of the optimum as a limit, provided an infinite running time, has been proven (see e.g., [Bäc96]). Similar facts are known for algorithms like Simulated Annealing. However, depending on the cost distribution, evolutionary algorithms can very well profit from restarting, simply because of the cost distribution's influence on the initialization.In case of a type-A distribution, the impact of re-runs may greatly improve the results, whereas in a type-B1 scenario, restarts do not make much of a difference. The influence is even weaker in case of a type-B2 problem. With type-B distributions, the results usually do not justify the higher costs in terms of running time.

In Table 1, the basic tendencies of influence are summarized. The three types of cost distributions directly suggest three classes of difficulty—from an evolutionary algorithm point of view. Type-A is the easiest,

where all components but recombination are positively influenced by the distribution. The impact on Type-B1 problems is fairly balanced; in Type-B2 problems negative influences dominate.

4 CASE STUDY

To corroborate our analysis, we scrutinize three representative and well-understood NP-complete optimization problems: *Set Partioning*, *Knapsack*, and *Traveling Salesman* Problem (see e.g. [GJ79]). The three problems are archetypical optimization problems which have been receiving great attention ever since their inception.

As a preliminary study to the experiments presented below, we scrutinized the occurring cost distributions with respect to the variance among instances of the same problem. For all three types of problems we varied all available parameters like size, type and parameters of the distributions of weights or coordinates etc. and determined the cost distributions with large uniform samples.

We found the cost distributions of our three problems converge very quickly (i.e. for virtually all non-trivial problem sizes) to their anticipated analytical continuous approximation, according to the Central Limit Theorem. This statistical analysis, based on the structure of the problem and its cost function, is presented in [Waa99]. For TSP, we have also determined the cost distributions of all problems given in the standard benchmark library TSPLIB [Rei91, WGL00b], and found them to match the same characteristic shape.

In this section we will present data taken from a larger series of experiments in which we scrutinized the effect of the cost distributions. To ensure the results of the experiments are comparable across the different problems, we implemented a generic framework which provides a uniform way of controling common parameters like the ratio of individuals generated by recombination to the those stemming from mutation etc. All experiments below were conducted using 250 individuals per generation; the optimization was terminated after 1000 generations.

While writing our own framework guarantees a fair analysis, we compared our findings with results in the literature verifying the generality of our observations.

4.1 TYPE-A

Set or number partioning is a typical representative of this class. A set S of numbers is to be partitioned into

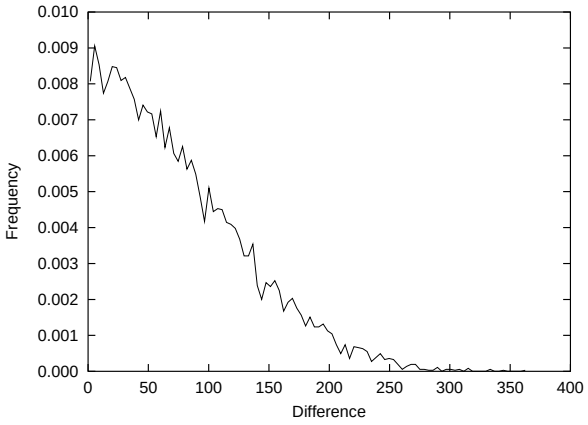


Figure 3: *Set Partitioning: Cost distribution for problem instance of size 150*

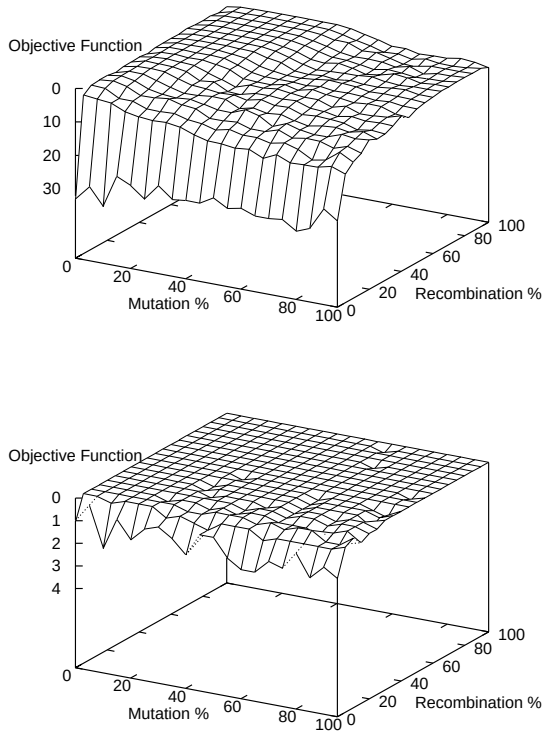


Figure 4: *Set Partitioning: Quality of optimization result as function of mutation and recombination ratio; z-scale inverted, smaller values indicate better results (optimum at 0). Population size 100 (above), 1000 (below)*

2 subsets S_1 and S_2 such that $S_1 \cup S_2 = S$, and the difference of the sums

$$|\sum_{s \in S_1} s - \sum_{s \in S_2} s|$$

is minimal.

Figure 3 shows the cost distribution of an instance with 100 elements taken from a Gamma distribution. The original distribution of numbers decreases in significance with increasing size of the set. For an analytical model, we refer the reader to [WGL00b]. The cost distribution is characterized by optimal and near-optimal costs appearing with the highest frequencies. Even plain random sampling algorithms are guaranteed to find near-optimal solutions [KKLO86], hill climber and other multi-start algorithms that do not deploy highly sophisticated techniques, achieve excellent results within extremely short running time.

Evolutionary algorithms find results of similar quality but require longer running times. With this kind of distribution, the size of the initial population is critical to the stability of the optimization, i.e., using a population size of 1000 almost certainly contains an optimal or near-optimal solution; the quality of small populations may differ significantly, so that using a tight time limit and re-starting the algorithm a couple of times may improve the results significantly in case of small populations.

We implemented a genetic algorithm for set partitioning using the standard string encoding. Figure 4 shows the dependencies between the quality of the optimization result and recombination and mutation ratio respectively. Besides the ratio of mutation and recombination, we also varied the size of the population.

- The figure underlines the importance of the initialization (value for (0,0) represents plain random sampling): Using a large populations we obtain near-optimal results without recombination or mutation
- We achieve (near-)optimal results easily for almost any configuration
- No sophistication is needed when defining the recombination operation

4.2 TYPE-B1

The class of type-B1 distributions comprises, among others, a large variety of scheduling, timetabling, assignment problems, and *Knapsack Problems* on which we focus here.

The problems definition is as follows: Given a number of items—each has a profit and a weight associated with it—, a (sub-)set of items is sought such that the total weight does not exceed a given bound but the sum of profits is maximal (see e.g., [GJ79]). We inverted the values to turn the maximization problem in one of finding the minimum. In Figure 5, the cost distribution of an instance consisting of 150 items is shown. The values of both weight and profit of the single items were chosen as random numbers between 10 and 100. The capacity of the knapsack was chosen as half the total weight of all items. Such assumptions are common in the literature [ZT99]—in particular, this configuration follows the example of [MT90].

The effect of this distribution on genetic search is twofold: The sampling of an initial population does not contain high quality solutions. Also, the random sampling component within cross over and mutation is limited—the probability to sample a near optimal solution is practically zero. On the other hand, the optima are not too far away from the bulk of solutions. Genetic algorithms are known as a suitable and very successful optimization technique for this kind of cost distributions.

Figure 6 shows the experimental results obtained with an implementation using standard string encoding of individuals.

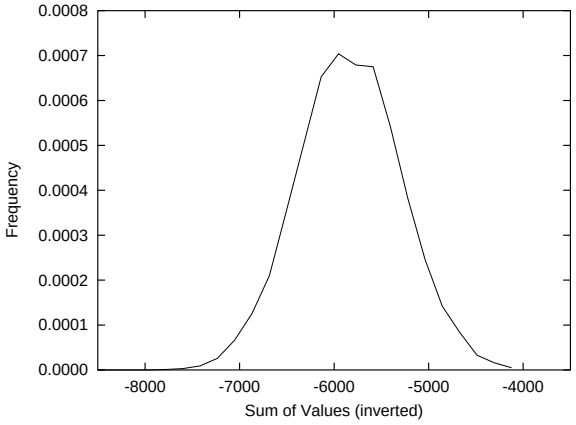


Figure 5: *Knapsack Problem: Cost distribution for problem instance of size 150; x-scale negated; (optimum at -8386)*

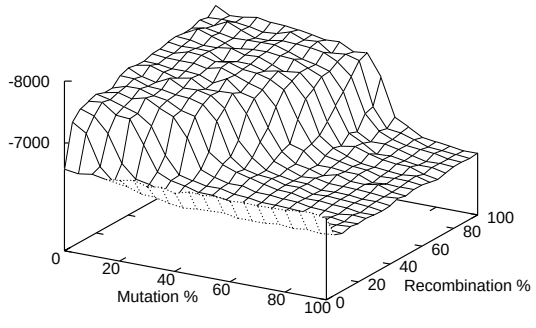


Figure 6: *Knapsack Problem: Quality of optimization result as function of mutation and recombination ratio*

- In contrast to type-A problems, the population size is of little importance (we omitted figures for experiments with different population sizes as they are identical)

- With an increasing ratio of recombination, the result quality improves–we see significant improvements over random sampling and high mutation ratio; best results are obtained with high recombination and low mutation ratio (about (80%,20%)).

4.3 TYPE-B2

As a Type-B2 problem, we study the cost distributions of the symmetric TSP where instances are given only by the coordinates of the cities. The TSPLIB collection of instances for the symmetric TSP serves as a widely accepted standard benchmark library in this field [Rei91]. In Figure 7 the cost distribution of a problem with 52 cities, obtained from 10^6 uniformly sampled tours, is depicted. The cost distribution shows the expected features: Almost all solutions are concentrated—even in the upper half of the total cost range. Moreover, they are concentrated in a very small interval. The optimal tour is known to be of length 7542. All sampled tours are longer than 21966 and shorter than 35898. Consequently, neither when randomly selecting tours for a initial population nor when adding randomly chosen tours during the optimization a tour shorter than 21966 is likely to be chosen. The best sampled tour is more than twice the length of the one found by a simple greedy algorithm (9535).

The TSP is know to be a difficult problem for evolutionary algorithms. Evolutionary algorithms when applied to this problem require special, sophisticated extensions in order to achieve competitive results (see e.g., [MW92]).

Figure 8 shows the results obtained with our implementation. We experimented with different recombination strategies found in the literature; while differing in result quality, the overall trends as depicted in the graph could be observed with all implementations. Mutation was implemented as 2- or 3-swaps.

- Both initialization and size of the population are virtually irrelevant, i.e. all random tours are significantly suboptimal and any population of non-trivial size, say, 100 or greater will lead to very similar results.

- Increasing the ratio of individuals generated by recombination leads to better result performance though results are suboptimal on average;

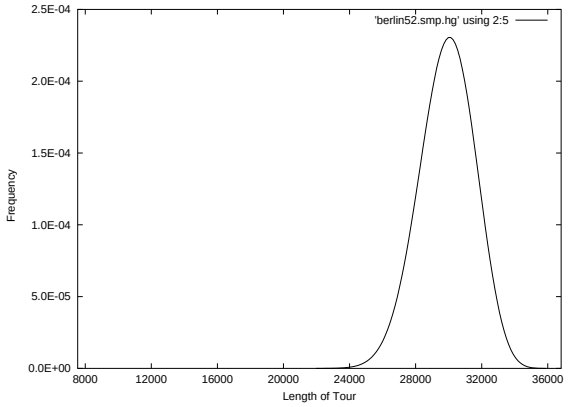


Figure 7: *Traveling Salesman Problem: Cost distribution for problem* berlin52 *of TSPLIB (optimum at 7542)*

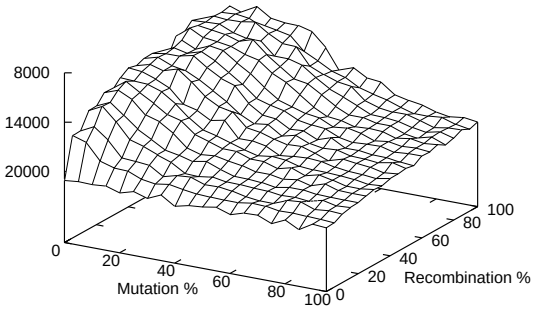


Figure 8: *Traveling Salesman Problem: Quality of optimization result as function of mutation and recombination ratio*

- Due to the low concentration of (good) neighbors, mutation can easily decrease the number of good or prospective individuals; as opposed to the other two problem types, the range of good results is smaller, around a low ratio of mutation only

- Restarts have practically no influence on the result quality.

Interestingly, we found the same trends with different recombination techniques.

5 SUMMARY

Based on the observation that a cost distribution of an optimization problem is characteristic for the problem [WGL00b], we studied its effects and implications on an optimization with evolutionary techniques. Cost distributions come in three major types of shape: a strong concentration of costs similar to the optimum (1); or else the bulk of solutions has costs either far (2) or very far (3) from the optimum.

Our analysis shows which algorithmic principles of evolutionary search are positively and which are negatively influenced by a particular shape of the cost distribution. Summing these partial influences up, we gave experimental evidence that cost distributions indicate whether a problem is (1) too easy for an evolutionary approach, i.e., evolutionary search is an overkill and simpler algorithms perform just as well; (2) of a difficulty which evolutionary techniques are typically well suited to tackle; or (3) a hard problem, where the standard repertoire of evolutionary implementation techniques achieve only mediocre performance.

Unlike previous work in this field we deliberately avoided the notion of landscape, because it is not intrinsic to the problem but artificially imposed on the space, intently or not, to allow the use of navigation algorithms. In contrast, cost distributions are entirely inherent to the problem, and independent of the optimization algorithm applied. Furthermore, we observed that cost distributions could predict the behavior of evolutionary algorithms, which do introduce and utilize a space topology. It appears that cost distributions are influential to the definition of landscapes, as the difficulty of shaping a certain landscape depends also on the number of available solutions of certain costs. For example, a landscape which is favorable for Hill Climbing optimization is significantly easier to define in case of a type-A distribution than is for a type-B.

Our analysis provides an indicator whether a given problem is difficult enough to be tackled with evolutionary algorithms; and which component of an evolutionary search technique to modify and tune, in case the results are not satisfying.

References

[Bäc96] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, Oxford, 1996.

[FJMO95] M. L. Fredman, D. S. Johnson, L. A. McGeoch, and G. Ostheimer. Data Structures for Traveling Salesmen. *Journal of Algorithms*, 18(3):432–479, 1995.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Co., New York, 1979.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, USA, 1989.

[Kau93] S. A. Kauffman. *The Origins of Order*. Oxford University Press, New York, Oxford, 1993.

[KKLO86] N. K. Karmarkar, R. M. Karp, G. S. Lueker, and A. M. Odlyzko. Probabilistic Analysis of Optimum Partitioning. *Journal of Applied Probability*, 23:626–645, 1986.

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, New York, USA, 1990.

[MW92] K. Mathias and D. Whitley. Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem. In *Parallel Problem Solving from Nature*, pages 219–228. Elsevier Science Publishers, 1992.

[Rei91] G. Reinelt. TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[Waa99] F. Waas. Cost Distributions in Symmetric Euclidean Traveling Salesman Problems—A Supplement to TSPLIB. Technical Report INS-R9911, CWI, Amsterdam, The Netherlands, September 1999.

[WGL00a] F. Waas and C. A. Galindo-Legaria. Counting, Enumerating and Sampling of Execution Plans in a Cost-Based Query Optimizer. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 499–509, Dallas, TX, USA, May 2000.

[WGL00b] F. Waas and C. A. Galindo-Legaria. The Effect of Cost Distributions on Genetic Algorithms. Technical Report INS-R0003, CWI, Amsterdam, The Netherlands, January 2000.

[WM97] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Trans. on Evolutionary Computing*, 1(1):67–82, April 1997.

[ZT99] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Trans. on Evolutionary Computing*, pages 257–271, November 1999.

Genetic Algorithms: Combining Evolutionary and 'Non'-Evolutionary Methods in Tracking Dynamic Global Optima

Simon M Garrett
Dept. Computer Science
University of Wales, Aberystwyth
Wales, SY23 3PG. UK
smg@aber.ac.uk

Joanne H Walker
Dept. Computer Science
University of Wales, Aberystwyth
Wales, SY23 3PG. UK
jhw94@aber.ac.uk

Abstract

The ability to track dynamic functional optima is important in many practical tasks. Recent research in this area has concentrated on modifying evolutionary algorithms (EAs) by triggering changes in control parameters, ensuring population diversity, or remembering past solutions. A set of results are presented that favourably compare hill climbing with a genetic algorithm, and reasons for the results are suggested. A method is then introduced, Evolutionary Random Search (ERS), that combines crossover and hill climbing mutation in a novel manner. It is assessed against the GA and hill climbing tests, and the encouraging results are discussed.

1 Introduction

There is a growing literature concerning the finding and tracking of *dynamically changing optima*, by evolutionary computation, e.g. Branke (1999a); Grefenstette (1999); Oppacher and Wineberg (2000); Eggermont et al. (2001); Ursem (2000). Almost all of these methods take the form of modified genetic algorithms (GAs), although some have compared GAs and Evolution Strategies (ES), e.g. De Jong (1999). None could be found that used hill climbing in dynamic optimisation. This paper provides an updated summary of the literature in this field and investigates the value of hill climbing versus GAs in tracking optima.

Four optimisation methods — two types of GA and two types of Random Mutation Hill Climbing (RMHC) — were applied to optimisation tasks of different types, and levels of complexity. The results agree with those of Mitchell et al. (1993) who found that RMHC was more effective than typical GA approaches. The work presented here suggest that this is also true when following *dynamically* changing optima.

Continuing from the comparative results, suggestions are made about how a new hybrid evolutionary-hill climbing method might be obtained, by combining elements from GAs and RMHC. Three versions of the new method, Evolutionary Random Search (ERS), are tested on the optimisation tasks. Conclusions are drawn from the results and suggestions are made about the relationship between ERS, RMHC and GAs.

2 Background

This section reviews previous work on: (i) comparisons of EAs and hill climbing, and (ii) evolutionary techniques for dynamic optimisation.

2.1 EAs and Hill Climbing

In 1993, Mitchell et al. (1993) asked the question, "When will a genetic algorithms outperform hill climbing?", in work that set out to test Holland's Building Block Hypothesis (1975). To this end an experiment was devised that, it was thought, would lay out a simple *Royal Road* for a GA to follow, from small schemas through to the optimal string, and compared it to three hill climbing methods. Instead it was found that one hill climbing algorithm, RMHC, outperformed the GA by an order of magnitude. RMHC was defined as follows, "In RMHC, a string is chosen at random and its fitness is evaluated. The string is then mutated at a randomly chosen single locus, and the new fitness is evaluated. If the mutation leads to an equal or higher fitness, the new string replaces the old string[1]. This procedure is iterated until the optimum has been found, or a maximum number of function evaluations has been performed", on a population of individuals.

[1] Note that this mutation operator is *not* destructive.

The main area of weakness in the GA was identified as a property called *hitchhiking*, that occurs when an unfit gene near to a fit schemas is spread through the population along with the fit schema. One of the reasons RMHC outperformed the GA was because the GA was doing wasteful work removing these hitchhikers, whereas RMHC was not.

Lang (1995) suggested that hill climbing improved on results in Koza (1992), beginning a series of claims and counter claims, involving Koza's informal rebuttal at the ML-95 conference and hearty discussions on the internet (Lang et al., 1995). Both Lang and Koza seemed to overlook Lang calling his hill climbing method 'RMHC' in their internet exchanges. According to Lang (1995), and the definition given above, the method used was *not* RMHC since it involved crossover, and apparently did not create new random solutions as mutations of previous solutions.

True RMHC can be considered to be a form of parallel evolution, akin to multiple, isolated (1+1)-ES populations, in which the population's single member competes only with its offspring for survival. However, in RMHC the degree of mutation is not usually under the control of strategy parameters.

2.2 EAs and the Dynamic Optimisation Problem

The motivation for our work is, however, quite different from Mitchell et al. (1993). It springs from a need to optimise robot behaviours in a changing environment, see Walker and Wilson (2002), and hence relates to a real-world problem in evolutionary dynamic optimisation.

Angeline (1995) and Branke (1999a) have summarised much of the work in this field, to which the reader is referred for details. However, relevant aspects of their papers, and additional, more recent work, are presented below. Branke usefully categorises this work as exhibiting one of the three following characteristics:

- Triggered change to the EA's operation when a change in optima is encountered.
- Continual diversity so the EA's population can always respond to changes in optima.
- Remembering previous solutions (e.g. previous global optima) to be reused later.

All three approaches aim to mitigate the lack of diversity in a standard, optimised EA population. When diversity is lacking, and the optima change, most (if not all) of the EA's search points can remain located around the old global extreme, and thus they can easily be trapped in new local optima near that point.

2.2.1 Triggered Change

The first approach is that of *triggered change*, an early example of which is Cobb's work in *triggered hypermutation* (Cobb, 1990), improved by Cobb and Grefenstette (1993).

Grefenstette and Cobb's approach triggered a large increase in mutation, when the environment changed, to increase the diversity of the population. The mutated individuals were able to search areas of the fitness space away from the old point of convergence and, by means of the crossover operator, the population could spread throughout the new fitness space, before converging again, at or near the new global extreme. The process can repeat indefinitely. Grefenstette has recently provided a comparison of various types of mutation and hypermutation models (Grefenstette, 1999).

Grefenstette pointed out that Cobb's triggered hypermutation fails in some types of dynamic environments (Grefenstette, 1992). Firstly, if the environment changes significantly and the new optima are not close enough to previous ones, hypermutation may not introduce enough diversity to overcome this. Secondly, if the fitness space changes only by *adding* new optima, then hypermutation would not be triggered at all. The solution involved a proportion of the population being continually replaced by random individuals, an approach called the *Random Immigrants GA*. In a comparison between the performance of a standard GA, triggered hypermutation and random immigrants algorithms, Cobb and Grefenstette (1993) found that in dynamic environments with large scale changes, random immigrants performed best, it also caused less disruption in stable environments.

The Random Immigrants method was further modified (Grefenstette, 1999), where an attempt was made to control the mutation rate genetically, in a similar manner to an ES. Grefenstette used this in a number of adaptive mutation rate tests, and found that, although techniques using some hypermutation gave better results than those that did not (for both gradually and suddenly changing environments), altering the mutation rate dynamically was not as effective.

2.2.2 Continual Diversity

Grefenstette's work using random immigrants might be thought of as a move towards *continually ensuring* high diversity, in place of regaining population diver-

sity when change is detected. This section discusses other means of maintaining continual diversity.

Ghosh et al. (1998) implemented an aging population of individuals because dynamic optimisation was considered to be, "...optimizing a series of time-dependent optima." This approach added to the Steady State GA (SSGA) such that each individual was given an age, used in calculating its fitness (along with performance metrics), with middle-aged individuals gaining the most fitness. It was found that his form of GA improved on the performance of the SSGA in both stationary and non-stationary environments. In dynamic environments the new version outperformed the SSGA for small environmental changes, and was particularly impressive for large scale changes.

Ursem's comprehensive work (Ursem, 2000) set out a relatively complex algorithm, which views the GA's population as a number of subpopulations that may be climbing different peaks in the fitness landscape. These subpopulations were identified and maintained during subsequent generations, for instance by chosing crossover mates from within the same subpopulation. It was found to outperform the sharing GA (Goldberg and Richardson, 1987).

Another project using subpopulations took a simpler approach (Oppacher and Wineberg, 2000), a main "core" population, and numerous "colonies" around it that explored different parts of the fitness landscape. These colonies were forced away from the core population's fitness space, and performed hill climbing to avoid unnecessary exploration of poor parts of the landscape. Crossover occurred within colonies and the core only. Good colony members could periodically migrate to the core, to provide it with increased diversity and to allow it to adapt quickly when the environment changed. It was found to outperform a standard GA in dynamic environments.

2.2.3 Remembering Solutions

If a new optimisation problem is similar to a previous solution, it might be more efficient to remember the old solution than to regenerate it. Again Branke makes a useful distinction here between two types of approach: firstly, *implicit memory* often using multiploidy, e.g. Dasgupta and McGregor (1992) and Lewis et al. (1998), although neither appears to be particularly robust; secondly, *explicit memory* where solutions are specifically stored for later reuse. With Ramsey, Grefenstette has addressed this issue too (Ramsey and Grefenstette, 1993), but this does assume that the environment can be measured, so that the relevant solution can be retrieved.

The method described in (Louis and Xu, 1996; Louis and Johnson, 1997) sampled and remembered the best member of the population at regular intervals. It was found that seeding the next run of the GA with 5-10% of the remembered individuals gave improved results. However the method appeared to be fragile to a higher percentage of seeding, and to large changes in the fitness space.

Eggermont et al. (2001) have reported a GA with a case based memory of past successes, which the GA accessed when the environment changed. After each generation, the best individual was added to the memory, then when the fitness deteriorated, e.g. due to environmental change, the individuals in memory were re-evaluated and the best in the current environment was re-introduced into the population. It was found that this case-based addition to the GA improved the performance in a dynamic environment.

Branke (1999b) himself describes a method that defined two populations: a memory population to remember previous, good solutions, to maintain a minimum degree of quality; and a population to search constantly for new peaks, submitting its best efforts to the memory population. The search population was reinitialised after every change in the fitness space.

3 Aims and Objectives

In light of the research, just reviewed, the aim of this work is to answer two questions: "Can RMHC and GAs be combined to produce a method that ensures a continually diverse population?" and, "How will such a method perform, relative to RMHC, and why?" The objectives to obtain these aims are two-fold:

Firstly, to compare the performance of GA and RMHC methods in a dynamically changing, multidimensional environment, under a variety of conditions. The different conditions should test different aspects of the methods and highlight their benefits.

Secondly, to use this information, and previous work in evolutionary dynamic optimisation, to begin to generalise a new method of optima-tracking. This method should have a performance which approaches or exceeds RMHC, and which improves on standard GA performance.

4 Methodology

4.1 Optimisation tasks

The optimisation tasks to be solved were 2-, 5- and 10-dimensional versions of Equation 1. Increasing the

number of dimensions, from 2, to 5 to 10, made the optimisation task more difficult, since this reduced the density of search points in the fitness space. In all cases the space searched by the methods below was limited to an $0 \leq x_i < 10$ square/hypercube. The standard benchmark equation to be optimised was,

$$f(\mathbf{x}) = \frac{sin(5(\sum_{i=1}^{N}(x_i - xf_i)^2)^{1/2})}{5(\sum_{i=1}^{N}(x_i - xf_i)^2)^{1/2}} \qquad (1)$$

for $N \in \{2, 5, 10\}$, and where f defines the fitness at a point in space represented by the genes of an individual[2], with $f \to 1$ as the Euclidean distance from the origin to $\mathbf{x} \to 0$.

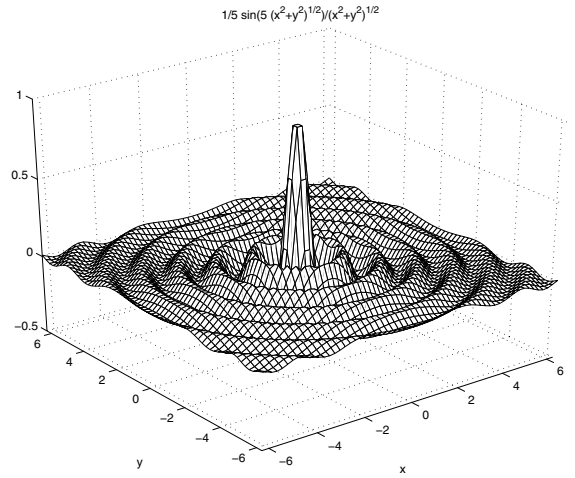


Figure 1: A 2-D graph of Equation 1, here the global maximum is at (0,0)

To implement the dynamics required, the global optimum (in this case a maximum) was relocated by adding an offset, xf_i, to each x_i. A change occurred every 1000 generations, twenty times. The location of the maximum changed in two ways:

- Randomly within the space [(0,0),(10,10)). This tested the ability of the algorithms to find optima that were often well-separated in the fitness space, with the complication of intervening local optima. Methods with poor continual population diversity should not perform well.
- Incrementally from (0,0) to (2,2), in 0.1 increments. This tested the ability of the algorithm to track optima that were relatively close in the fitness space. In some cases, parts of the maximal peak were outside the search space. This made the search task slightly harder than if the maximum were always totally with the space.

[2] Refers to both GA and RMHC population elements

4.2 Comparative Tests: the GA and RMHC Methods

The methods used were:

GA An unremarkable implementation, using two-adversary tournament selection. Crossover swapped a two real-valued genes, at a random locus, with a probability of 0.8. The mutation operator, which set the new value of a gene randomly in the range [0,10), had a probability of mutation of 0.01. The GA's control parameters were adjusted by hand to maximise its performance.

GA with Elitism (GA-E) Implemented as described in the previous point, except for the addition of elitism, which retained the best member from each generation.

Global mutation RMHC (RMHC-G) Both forms of RMHC followed the definition in Mitchell et al. (1993), set out above. The only variation was in the type of mutation. RMHC-G used the GA's mutation operator and set a 'gene' in the range [0,10). Note that there was no need to maximise the performance of either form of RMHC.

Local mutation RMHC (RMHC-L) Identical to RMHC-G, described in the previous point, except that the RMHC-L mutation operator ensured that a new 'gene' value, v' was close to the old value v. Given r, the range of possible values for v, this was achieved by, $v' = v + random() * r/100 - r/200$. Unlike the GA, both forms of RMHC are only able to alter one dimension (gene) before having fitness evaluated. This placed them at a disadvantage to the GA methods, in which crossover can alter several dimensions at once.

All methods used real-valued chromosomes, not bit strings, with a population size of 100, and 1000 generations of continuity. Tests showed that raising these levels by an order of magnitude made very little difference in relative performance of the methods, and lowering the values by an order of magnitude only slightly changed their relative performance. Since 100 generations and populations of 10 make the results less repeatable, the values above were preferred and the optimisation tests were run under these conditions for all four methods.

4.3 ERS: Improving on the Standard Methods?

Another approach, introduced here, combined elements of these algorithms to form an *Evolutionary Random Search* (ERS) method. In population terms, an ERS is analogous to removing a number of individuals from a population and shipwrecking them on an island, where they and their offspring compete against each other. During this time they may be considered a subpopulation, as discussed Section 2.2.2, however they do not necessarily represent points near local optima. The ERS *framework* (Figure 2) allows instances of an ERS to vary the number of shipwrecks that occur, and the number and types of offspring. In all cases however, a shipwreck involves some combination of RMHC's non-destructive mutation and GA's crossover. There are t shipwrecks per generation. Three versions of ERS are discussed here. In all three examples, $n = 3$, and P, the population of individuals, had 100 members; but this need not be the case.

- Repeat t times:
  - Select n members, **s**, from P.
  - Create m_1 RMHC-G mutants from **s**.
  - Create m_2 RMHC-L mutants from **s**.
  - Select members from the local and global mutants.
  - Crossover selected individuals and create children.
  - List originals, mutants and children.
  - Sort list.
  - Place fittest n from list back into P.
- End repeat.
- Replace any remaining individuals in P with random (or remembered) individuals.

Figure 2: The ERS framework (one generation)

The ERS1 method combined GA, RMHC-G and RMHC-L in a deliberately naïve and expensive manner. Its purpose was to show the combined effect of GA, RMHC-L and RMHC-G. During each generation there were $|P|/n$ shipwrecks, covering the whole of the population, bar one individual. This individual could be replaced with a random individual, or with a remembered previously successful individual, to help the recovery of previously seen solutions. For these tests the individual was randomised. Each iteration of the repeating portion of Figure 2 created 12 new individuals. Three of these were global mutants, three were local mutants and 6 were the offspring of the two groups of mutants. Thus the naïvety of ERS1 was due to it being *four times more expensive* than RMHC and the GA.

ERS2, was identical to ERS1 in all but two respects. Firstly, it created only 8 shipwrecks per generation, to ensure nearly the same number of fitness evaluations (96) as the GA and RMHC methods (100), making it possible to compare the effectiveness of ERS2 and RMHC-G. Secondly, it randomly chose the members of its 8 shipwrecks, and the remaining individuals were left unaffected for that generation.

ERS3 tried a different combination of parameters to expore other possibilities of the ERS framework, whist again performing the same number of fitness evaluations as the ERS2, GA and RMHC methods. ERS3 ordered the population by fitness, and split it into three parts of $|P|/3$ individuals (the remaining individual was replaced by a new random individual.) It selected an individual from each of the three parts, called *hi*, *mid* and *lo*, to indicate their relative fitness. Each shipwreck in ERS3 applied crossover, and non-destructive mutation to these three members, in a manner that maximised the chance that the offspring would represent improvements to the optimisation task. To this end, ERS3 dispensed with RMHC-L (i.e. $m_2 = 0$), since the results below will show it provided little in terms of performance, and because RMHC-G can do local optimisation where required, although it takes longer. Both of the RMHC-G mutants were of the *hi* individual because, with no other information available, each was more likely to lead to a fitter individual than mutants of *lo* or *mid*. These two new individuals are called $gm1$ and $gm2$. The remaining individual was created by crossover of *lo* and *mid*, in the hope of occasionally combining two poor solutions into one fit one. This operation also simultaneously altered multiple dimensions, something the mutation operator can not do. One of the two offspring was retained at random to give xi. At this point the fitnesses of $gm1$, $gm2$ and xi were evaluated. The three best individuals, of the six total individuals on the island, were then returned to the population.

5 Results

Figure 3 shows the best and average fitness, for each generation of a GA tracking a 2-dimensional version of the randomly changing function. Since the best and average fitness were closely related in all tests, the average fitness results will be omitted in the following results. The full results set can be found at `http://www.aber.ac.uk/`

smg/WORK/ga-hc-results.htm, or on request. They consist of the best, average and worst fitness for each generation of each experiment, and re-runs of the full set of results to demonstrate repeatability, as well as results for other fitness landscapes.

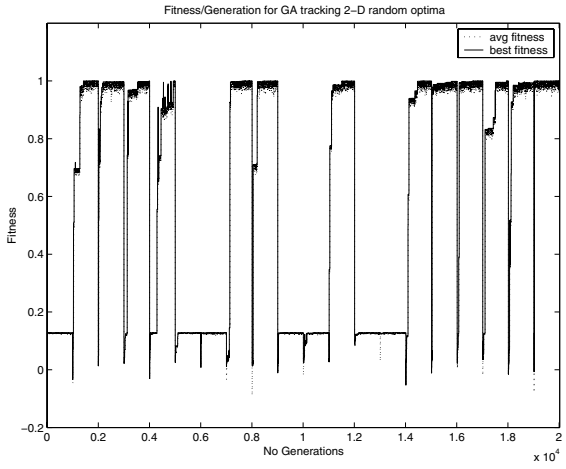


Figure 3: A typical task: the GA optimising a 2-D, randomly changing fitness function

Figure 3 illustrates how a population's best fitness dropped every 1000 generations, when the fitness function's optima were changed. The fitness then often returned to previous levels. At times, however, the GA could not find the global maximum in the allotted number of generations because all population members were caught around a local maximum, and mutation was not occurring often enough to provide search points that were fitter than the local maximum. The average best fitness is used to summarise the success of each of these tests, as shown in Table 1. The standard deviation is also available in the full results.

The test results from Table 1 are presented in Figures 4 and 5. These show the two types of dynamic test, random (left) and incremental (right), for 2-, 5- and 10-dimensional versions of the test function.

5.1 Comparative Test Results

The incremental results for RMHC-L, the GA and the GA-E were fairly dependent on the fitness of the initial population — when high, the optimisation method might track the maximum for a number of generations. However, there were some clear results.

In all cases RMHC-L performed worst, with the exception of the 10-dimensional incremental test, an artefact of the RMHC-L's initial population in the result presented here. The GA and GA-E performed better;

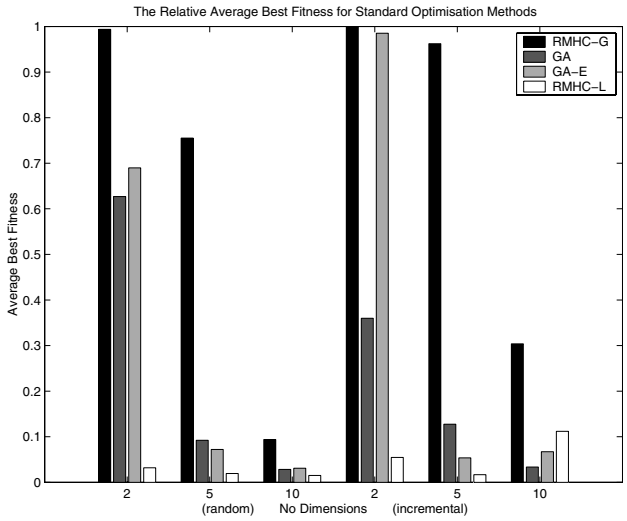


Figure 4: The comparative results: the left-hand group of three plots show the randomly moving optima results and the right-hand group show the incrementally moving optima results, both for 2-, 5- and 10-dimensions

almost equally as well as each other, except for the 2-dimensional incremental test at which GA-E was superior. RMHC-G was by far the most successful method. These results indicate that the results in Mitchell et al. (1993) apply to dynamic optimisation, as well as to static optimisation, under some conditions.

The two types of RMHC performed best and worst, indicating a relationship between the amount of non-destructive mutation and utility. Presumably an ES (without crossover) would perform somewhere Between these two extremes, since Gaussian mutation can result in both large and small changes. This will be tested in future work.

5.2 Evolutionary Random Search (ERS) Test Results

Figure 5 shows that ERS1 was better that RMHC-G in all cases. Since ERS1 performed the same amount of random mutation as RMHC-G, plus extra RMHC-L and crossover operations, this is not surprising. What is perhaps unexpected is that ERS1 does not out-perform RMHC-G by much, except in the 10-dimensional incremental tests. However, when the ERS method is limited to the same amount of processing as other methods (ERS2 and ERS3), the difference in performance between them and RMHC-G was not as clear (although both types were still superior to GA-E, GA and RMHC-L).

Table 1: Data for Figures 4 and 5

	Random			Incremental			
No. Dimensions	2	5	10	2	5	10	Average
ERS1	0.995	0.774	0.152	0.999	0.995	0.977	0.8157
ERS2	0.986	0.224	0.075	0.998	0.989	0.958	0.7056
RMHC-G	0.993	0.755	0.093	0.998	0.962	0.303	0.6846
ERS3	0.990	0.517	0.068	0.998	0.957	0.114	0.6077
GA-E	0.689	0.072	0.031	0.985	0.053	0.067	0.3165
GA	0.626	0.092	0.028	0.359	0.127	0.033	0.2113
RMHC-L	0.031	0.019	0.015	0.054	0.016	0.111	0.0415

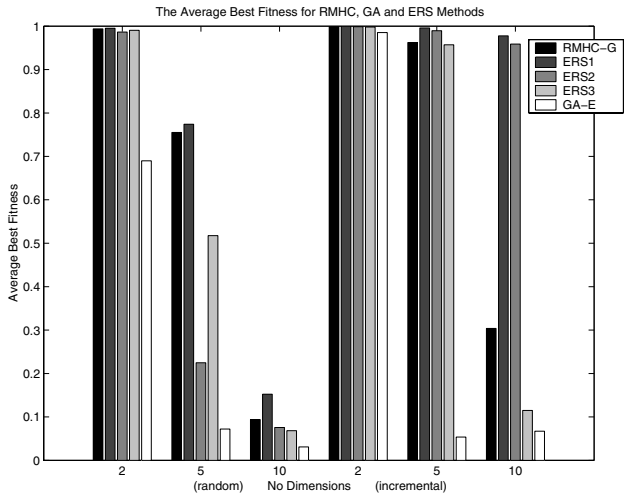


Figure 5: A comparison of the ERS methods to the GA-E and RMHC-G methods

To assess overall performance the average of each row in Table 1 was taken. The most successful was ERS1, with an average fitness of 0.8157, though its result can be dismissed due to the excess fitness evaluations mentioned above. This was followed by ERS2 with a fitness score of 0.7056, closely followed by RMHC with a score of 0.6846 (the difference is not statistically important). ERS3 performed respectably with 0.6077, and the best of the remaining methods, GA-E, had an average fitness of 0.3165.

Overall RMHC-G and ERS2 could hardly be separated in terms of performance, and this suggests that there is scope for further research into combinations of evolutionary and non-evolutionary methods. The ERS methods also showed significant improvement over the GA methods, due to their non-destructive mutation operators promoting continual population diversity (with *no need* to trigger some form of hypermutation). Comparing the ERS3 and RMHC-G results is interesting since it suggests that crossing over two poor individuals in ERS3 regularly yielded an individual that was almost as fit as a globally mutated individual in RMHC. Further work with different types of crossover is needed to establish whether this conclusion holds in general, or at all. If so this may indicate that ERS3's combination of non-destructive mutation, and weaker GA crossover, partially mitigates the hitchhiking phenomenon Mitchell et al. (1993).

6 Conclusions

As far as the authors are aware, the work presented here is the first attempt to unify evolutionary and non-evolutionary methods in dynamic optimisation. The first aim of this work was to answer: "Can RMHC and a GA be combined to produce a method that ensures a continually diverse population?" In this instance it can, by means of continual high levels of non-destructive mutation. The question now is how generalisable these results are.

The second aim was to discover how ERS performed relative to RMHC, and why. The results were good, but mainly because, like RMHC, ERS contains such high mutation. Perhaps this was to be expected since Grefenstette and Cobb noted that random immigrants were useful, but controlling the rate of their mutation had little value (Grefenstette, 1999); and since, in the results above, low levels of mutation were not particularly effective. The application of crossover, even to medium fitness chromosomes, appears to be quite effective, since it alters the values of several dimensions at once – something the mutation operator can not do.

On the basis of the results presented here, a suggested answer to Mitchell *et al*'s question, "When will a GA outperform hill climbing?" is "when it combines crossover with strong non-destructive mutation operators." In any case, EA practitioners — in contrast to those who model genetic and evolutionary processes — should neither dismiss EAs as inferior to RMHC, nor look for failings in the RMHC methodology above that will excuse the GA's performance. More fundamentally perhaps, one might question the need for a separation between EAs and hill climbing at all. Some

forms of ES are little more than self-adapting mutation; is this an EA or a form of hill climbing? Is such a dichotomy useful?

7 Further Work

The work presented here is being expanded by:

- Application and comparison of the methods to several, qualitatively different fitness functions, *c.f.* Ursem (2000).
- Comparison to other existing methods, such as ES and the approaches outlined in Section 2.2, particularly the use of age (Ghosh et al., 1998).
- Establishing whether the conclusions hold in general, and if so whether crossing over poor solutions can be used to mitigate the hitchhiking problem.
- Reducing the number of generations before the fitness function changes, until a change occurs every generation and testing the use of memory in the ERS (as briefly described in the text above).

References

Angeline, P. J. (1995). Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, Piscataway, NJ.

Branke, J. (1999a). Evolutionary approaches to dynamic optimization problems – a survey. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 134–137.

Branke, J. (1999b). Memory enhanced evolutionary algorithms for changing optimization problems. In *Congress on Evolutionary Comutation CEC'99*.

Cobb, H. (1990). An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time dependent nonstationary environments. NRL Memornadum Report 6760 (NCARAI report AIC-90-001).

Cobb, H. and Grefenstette, J. (1993). Genetic algorithm for tracking changing environments. In *Proceedings of the 5th International conference of Genetic Algorithms*, pages 530–532.

Dasgupta, D. and McGregor, D. (1992). Nonstationary function optimisation using the structured genetic algorithm. In *Proceedings of Parallel Problem Solving from Nature (PPSN-2)*.

De Jong, K. (1999). Evolving in a changing world. In *Foundations of Intelligent Systems*, pages 512–519.

Eggermont, J., Lenaerts, T., Poyhonen, S., and Termier, A. (2001). Raising the dead; extending evolutionary algorithms with a case-based memory. In *Proceedings of Second European Conference on Genetic Programming (EuroGP'01)*.

Ghosh, A., Tstutsui, S., and Tanaka, H. (1998). Function optimization in nonstationary environment using steady state genetic algorithms with aging of individuals. In *IEEE International Conference on Evolutionary Computation*, pages 666–671.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings 2nd International Conference on Evolutionary Programming*, pages 296–307.

Grefenstette, J. (1992). Genetic algorithms for changing environments. In *Proceedings of Parallel Problem Solving from Nature 2*, pages 137–144.

Grefenstette, J. (1999). Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In *Proceedings of the Congress on Evolutionary Computation (CEC99)*, pages 2031–2038.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor,MI.

Koza, J. R. (1992). *Genetic Programming*. MIT Press, Cambridge, MA.

Lang, K. (1995). Hill climbing beats genetic search on a Boolean circuit synthesis problem of Koza's. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 340–343, San Francisco, CA. Morgan Kaufmann.

Lang, K., Koza, J. R., and Tahoe, J. K. (1995). Webpages provided by J. K. Tahoe. http://www.genetic-programming.com/jktahoemain. html.

Lewis, J., Hart, E., and Ritchie, G. (1998). A comparison of dominance mechanisms and simple mutation on nonstationary problems. In *Parallel Problem Solving from Nature – PPSN V*, pages 139–148, Berlin. Springer.

Louis, S. J. and Johnson, J. (1997). Solving similar problems using genetic apgorithms and case-based memory. In *7th International Conference on Genetic Algorithms*, pages 481–488.

Louis, S. J. and Xu, Z. (1996). Genetic algorithms for open shop scheduling and re-scheduling. In *International Conference on Computers and their Applications*, pages 99–102.

Mitchell, M., Holland, J. H., and Forrest, S. (1993). When will a genetic algorithm outperform hill climbing. In *Advances in Neural Information Processing Systems*, volume 6, pages 51–58. Morgan Kaufmann Publishers, Inc.

Oppacher, F. and Wineberg, M. (2000). Reconstructing the shifting balance theory in a GA: taking Sewall Wright seriously. In *The Proceedings of the 2000 Congress on Evolutionary Computation (CEC2000)*, pages 219–226.

Ramsey, C. L. and Grefenstette, J. J. (1993). Case-based initialisation of genetic algorithms. In *5th International Conference on Genetic Algorithms*, pages 84–91.

Ursem, R. K. (2000). Multinational GAs: Multimodal optimization techniques in dynamic environments. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 19–26.

Walker, J. H. and Wilson, M. (2002). How useful is lifelong evolution in the physical world. Technical report, University of Wales, Aberystwyth. UWA-DCS-02-040.

A Genetic Algorithm with Self-Distancing Bits but No Overt Linkage

William A. Greene
Computer Science Department
University of New Orleans
New Orleans, LA 70148
bill@cs.uno.edu
504-280-6755

Abstract

We present a novel representation and crossover operator for genetic algorithms. Bits are not linked to one another. Instead, the current population suggests a pseudo-distance between each pair of bits; this pseudo-distance really measures the degree to which bits appear to participate in a building block. Then crossover respects the pseudo-distance: it is clumps of nearby bits that have their values copied from parent to child. Thus our new approach does directly (preservation of building blocks under crossover) what other approaches only hope to do indirectly. Our approach is tested on several problems, ranging from simple to very challenging, and the results compared to standard approaches. In these problems, the new approach is successful and usually outperforms the standard approaches.

1. INTRODUCTION

We assume there is a problem of interest to us, and we wish to use a genetic algorithm to search among the solutions to the problem. There is a plenitude of solutions to the problem, of varying quality; some are rather good solutions, and some are only fair. We assume there is a known measurement of the quality of a solution, which we term its *fitness* and which is a non-negative real number. Individual solutions are identifiable with the property values they exhibit, along a known set of properties. Solutions differ one from another by having different values for these properties.

In the standard representation used for genetic algorithms, an individual solution (which we now begin to term simply an *individual*) gets represented by representing its property values as values, which in this paper we will take to be bits, which are linked together in a linear sequence, like beads along a strand, that is, like genes along a chromosome. Under this representation, which mimics a biological model, mating with crossover continues the mimicry, in particular of haploidal reproduction. One or more crosspoints are chosen at random along the strand, parental genetic sequences are clipped at those points, and parental genetic fragments are exchanged, to form the children. Also mutation is easily mimicked, by changing an occasional bit value.

Two bits, as a pair, can assume any one of four bit-pair values, namely, (0, 0), (0, 1), (1, 0), and (1, 1). Let us say that two bits are *closely related*, provided that they can exhibit a bit-pair value that is rather beneficial to a solution, in the sense that having this pair of values significantly increases the fitness of the solution. This notion of a beneficial bit-pair value is an instance of what [Holland, 1975] terms a building block.

When we read the proof of the Schema Theorem (see [Holland, 1975] or [Goldberg, 1989]), we learn that two closely related bits can suffer from a great hazard. To lie far apart from one another along the bead strand increases the likelihood that the beneficial bit-pair value will be destroyed under crossover. One parent may exhibit the beneficial bit-pair value, but if a crosspoint is chosen between the pair of bits, it can happen that neither child exhibits the pair.

This hazard was recognized at the dawn of genetic algorithms, and since then attempts have been made to contend with it. One possibility is reordering bits, with the intention of having closely related bits wind up situated near one another. [Holland, 1975] noted that the *inversion* operator (a subsequence of bits gets its order reversed) might be useful in reordering bits dynamically. [Goldberg, 1989, pp.166-179] argues that for permutation-based representations (as might be used in the traveling salesman problem), certain permutation-based crossover operators, such as PMX [Goldberg and Lingle, 1985], combine the actions of crossover and reordering. The messy genetic algorithm mGA of [Goldberg, Korb, and Deb, 1989], along with its other distinctive features, implicitly permits reordering of bits. Bui and Moon have a sequence of papers that deal with ordering and preordering bits; of particular interest to them are graph problems, such as the graph bisection problem (see section 3.5 below). In [Bui and Moon, 1993] there is a preprocessing step which makes the sequence of bits (one for each graph vertex) reflect certain vertex adjacencies as they are evidenced in, say, breadth-first traversal. Then in [Bui and Moon, 1995] they follow a different course; this time the bits are placed, not in a one-dimensional sequence, but

instead at integral points of multi-dimensional real space; they argue the latter allows room for a more faithful representation of adjacencies. In [Greene, 2000] it is shown that under reasonable assumptions, a schema theorem obtains when the structure of bits and their linkages is liberalized to be as general as a connected graph, and then in [Greene, 2001] there are experiments with such alternative bit arrangements. [Sehitoglu and Ucoluk, 2001] explicate a regime for exchanging bits with their neighbors to the left or right, based upon whether they appear to participate in a building block.

(A new school, of probabilistic modeling, takes a completely different tack, but with the same general goal of identifying and exploiting those bits which are closely related. This school is exemplified by the research in [Muehlenbein and Paass, 1996], [Pelikan, Goldberg, and Cantu-Pas, 1998], and [Harik, 1999]. In this school, simulation of biological crossover is abandoned, in favor of a generate-and-test approach. The typical regimen is to loop on two steps: use the current population to infer some probabilistic dependencies between the values of the various bits, then use those probabilities to stochastically manufacture plausible individuals that will comprise the next generation.)

In the current paper, we will take a novel step towards correcting the hazard mentioned above. We will stay within the tradition that simulates crossover, but our bits will not be overtly linked together at all!

Reading the proof of the Schema Theorem suggests the following line of reasoning. When parental genetic material is inherited, it should be inherited in a certain piecemeal way. When parental bit values are copied into child bits, closely related bits should be copied in clumps. We will loop to copy parental bit values. When an uncopied parental bit is chosen to be the next one copied, we will copy not only that bit's value, but also the values of those still uncopied bits which are suitably closely related to it.

Above we alluded to reordering efforts, and non-linear linkage schemes. We now describe these as follows. Such approaches link or re-link bits one to another, with an eye to positioning closely related bits close together (generally this means a short path length between them). Then it should follow that when links are chosen for clipping during crossover, there will be a decreased likelihood that closely related bits get separated from one another.

The appeal of our approach is that it does directly what the other approaches only hope to do indirectly: closely related bits tend to clump together when parental genetic material is being copied into children. Linking bits one to another is a held-over artifact from the biological model of a chromosome (especially this is so when the linkage is into a linear sequence), and we now dispense with it.

Below we introduce a plausible measurement for close relatedness between bits. We will think of this measurement as also giving a pseudo-distance between bits. Our genetic algorithm will be generational (an entire new population $P(t+1)$ at time tick $t+1$ is created from the population $P(t)$ at time t, as opposed to a steady-state approach). Also, the measurement of close relatedness (or pseudo-distance) between bits is re-calculated for each generation. We term our approach a linkless self-distancing genetic algorithm.

2. THE LINKLESS SELF-DISTANCING GENETIC ALGORITHM

In this section we describe the details of our approach, LSDGA. Of course, our approach is mostly distinguished by a new way of representing an individual in the population, then by the algorithm for crossover that follows from it.

2.1. AN INDIVIDUAL

An individual is a set of bit-values. The bits are not linked together. Each individual in the population consists of the same number of bits, and that number is constant over all generations of the population. An individual has a fitness, which is a non-negative real number. The fitness value is problem-dependent.

2.2. CLOSELY RELATED BITS

How can we gauge when two bits are closely related? We do not purport to provide a perfect answer to that question, but our answer is plausible and persuasive. Future research may improve upon our answer.

As a pair, two bits can assume any one of four different bit-pair values, namely, (0, 0), (0, 1), (1, 0), and (1, 1). Our sentiment is that two bits are closely related, provided that they can exhibit a bit-pair value that is rather beneficial to an individual, in the sense that having this bit-pair value significantly increases the fitness of the individual. [Holland, 1975] would say that the two bits form a (small) building block.

We think of the current population as a sampling of the entire fitness landscape, and as such it offers evidence as to which bits are closely related. We consider just the fitter half of the current population (some other fractional part of the fitter individuals may be a better choice); denote this subset S. Now let two bits $b1$ and $b2$ be given. Considering just bits $b1$ and $b2$, the members of S may potentially exhibit any one of the four bit-pair values, and of course the members of S have their respective fitnesses. If among the members of S, fitness is rather disproportionately concentrated above just one of the four possible bit-pair values assumable by the bit-pair $b1$ and $b2$, then that is what we will deem close relatedness between $b1$ and $b2$. (Put another way, let us consider the opposite circumstance. If fitness is evenly distributed above the four bit-pair values assumable by $b1$ and $b2$, then we would say these bits are independent of one another, meaning that the value of one has little to do with the value of the other, as far as contributing to the fitness of an individual.)

Specifically, we do the following. For $i = 0, 1$, and $j = 0, 1$, let F_{ij} be the sum of the fitnesses of those elements of population subset S for which bit-pair ($b1$, $b2$) assumes the value-pair (i, j). Let $F = F_{00} + F_{01} + F_{10} + F_{11}$. The four fractions $r_{ij} = F_{ij} / F$ lie in the real unit interval [0, 1], and add up to 1.0. We want to detect the situation when one of these fractions is rather close to 1.0 (and the other three are nearly 0.0). More than one expression would reveal such; we use a familiar entropy calculation,

$$(1/2) \cdot \sum_{i,j} (-r_{ij} \cdot \log_2 r_{ij})$$

and denote this value by *dist*($b1$, $b2$). The normalizing factor 1/2 guarantees that this value lies in the real unit interval [0, 1]. Finally we observe that bits $b1$ and $b2$ more nearly fulfill our notion of being closely related exactly when *dist*($b1$, $b2$) more nearly approximates zero, and we will think of *dist*($b1$, $b2$) as being a pseudo-distance value.

At this point a reader may interject that a building block may consist of more than just two bits. We respond by reasoning as follows. If fitnesses are concentrated above one of the eight possible bit-triple values assumable by a triad of bits, then even more so is fitness concentrated above the bit-pair values of any two bits of the triad. This is because the latter concentration includes the concentration above the bit-triple value. On the other hand we also caution the reader as follows. For triads, the corresponding concentration metric (measuring the spread of eight fractions) should use not 1/2 but 1/3 as its normalizing factor. The concentration metric for a triad is not necessarily either greater or less than the concentration metric for a pair of bits drawn from the triad.

As remarked earlier, we practice generational evolution. Mating with crossover among sundry pairs of members of the population $P(t)$ at time t is used to create the next population $P(t+1)$. The first step in this generational rollover is to use (the fitter half of) population $P(t)$ to calculate the pseudo-distances between all pairs of bits; these distances will be used during crossover. Note that pseudo-distances are re-computed on each generation. (If an individual consists of 100 bits, there are 100 * 99 / 2 = 4950 bit-pairs, and so also that many distances get calculated.)

During crossover, we want bits that are suitably close together in pseudo-distance to get copied in clumps. To this end we calculate a *cutoff* value for pseudo-distance. We found the following worked well in our experiments. Bit-pair pseudo-distances are grouped into a histogram of twenty 5% brackets, then the chosen cutoff is that pseudo-distance that as an upper bound contains at least 20% of the bit-pair pseudo-distances (or sometimes it is 25%).

2.3. CROSSOVER

Crossover is now easily described. Two parents produce two children. To copy parental bit values into child bits, we loop on four steps. Step 1: choose an uncopied bit b at random. Step 2: to it, group those uncopied bits b' such that $dist(b, b') \leq \text{cutoff}$. Step 3: for k = 1, 2, copy this group of bit-values from parent(k) into child(k). Step 4: with 50% probability, interchange the roles of child(1) and child(2). (Step 4 means we may expect parts of a parent's genetic material to wind up in both children.) Let us note that the copying of isolated bits (ones not suitably close to other bits) resembles uniform crossover [Syswerda, 1989]. (In step 2, we sometimes prevent copying of an entire clump that is too large by making nearby bit b' jump a 50-50 hurdle before we group it with b.)

It is conceivable that premature convergence of the population can occur. Once the (fitter half of the) population members closely resemble one another, most bit pairs will seem to have fitness concentrated above a particular bit-pair value, so most bit pairs will appear to be quite close to one another, and a child may simply duplicate a parent. In section 2.4, which concerns how the next generation of the population is constructed from the current one, we take steps to diversify the population.

2.4 GENERATIONAL CHANGE

Our initial population $P(0)$ consists of random individuals.

To construct the next generation $P(t+1)$ of the population from $P(t)$, we first practice elitism, and have the fittest two members of $P(t)$ survive intact into $P(t+1)$. Then $P(t+1)$ is filled up to the same size as $P(t)$ by mating with crossover. As an aside, population size is kept small, typically between 25 and 50. Also, we linearly scale the set of fitness values present in the current population into a real interval of the form [1, maxVF] (for maximum virtual fitness) in such a way that the smallest fitness value in the population is mapped to 1, and the largest is mapped to maxVF. (Typically maxVF is 2 or 4.) Then, individuals are selected for parenting by using a weighted roulette wheel based upon the scaled fitnesses (see [Goldberg, 1989]).

Two parents produce two children. Each child is added to $P(t+1)$, but only if it is distinct from the individuals already in $P(t+1)$. Next we sort $P(t+1)$ into decreasing order of fitness, as a prelude to mutation.

The elite survivors in $P(t+1)$ are spared any mutation. For the rest, mutation is graduated and stochastic. A single mutation step consists of flipping the value of a randomly chosen bit. Each individual is subjected to some number of mutation attempts; in our experiments, the maximum number of attempts is 40. Mutation is stochastic, in that a mutation attempt succeeds to become a completed mutation step only with a 50% probability. Mutation is graduated, in that less fit individuals are subjected to more mutation attempts. The number of attempts is proportional to the rank of the individual within the (sorted) population. Thus the least fit individual is subjected to 40 mutation attempts, and we expect about 20 of these to result in the flipping of a bit in that least fit individual.

This completes the construction of $P(t+1)$.

2.5. STOPPING CONDITIONS

Generations of populations P(1), P(2), P(3), …, are formed, until some maximum number of generations has been reached, or some stopping condition has been met. Since we view genetic algorithms as a heuristic approach to problem solving, in this research we often content ourselves when a very good though sub-optimal solution has been unearthed. For many of the problems used in our experiments, a maximum fitness is known for the problem, and so we may content ourselves if we reach 98% of that value.

3. EXPERIMENTS

We have explored the behavior of our approach on a number of problems. The problems range from simple ones, to very challenging ones drawn from the literature.

3.1. COUNTING WEIGHTED 1'S

In this simple experiment, there are 80 bits, and they are numbered 1 through 80. The fitness of an individual equals the sum of the bit numbers of those bits whose value is 1 (versus 0). This problem is not quite the "counting 1's" problem (see Experiment 3.3); we might term it "counting weighted 1's". The maximum fitness is 1 + 2 + 3 + … + 80 = 80 * 81 / 2 = 3240. There is one individual of maximum fitness. The fitness landscape everywhere slopes upward towards this maximum. (Why? if we hill-climb in Hamming space, then changing a 0 to a 1 in an individual produces an individual of increased fitness, with the degree of increase depending on the bit number.) We used 98% of maximum fitness, or 3175.2, as acceptable fitness. We ran 20 trials of this experiment, using 40 as our population size, and allowing up to 100 generations per trial. We would expect the bits having high bit numbers to rapidly converge to the value of 1, and indeed this is what can be observed when a trial's generations are examined sequentially. All but two trials found an individual of acceptable fitness before exhausting all 100 generations. Over the 20 trials, the average fitness of the fittest individual found on a trial was 3186.75 and the average final generation number was 63.0. So, our new approach is successful at finding rather good solutions in a reasonable amount of time. Table 1 summarizes the results of this experiment.

Table 1: Counting Weighted 1's

Bits per individual	80
Optimal fitness	3240
Acceptable fitness	3175.2
Number of trials	20
Population size	40
Max generations	100
Avg final gen. num	63.0
Avg best fitness	3186.75

3.2. WEIGHTED 8-BIT GROUPS

This experiment is somewhat similar to the preceding one. Again an individual consists of 80 bits, but they are no longer numbered. Instead, they are grouped into 10 groups of eight bits each, and the groups are numbered 1 through 10. A subgroup of bits makes its own contribution to fitness. For group number k, $1 \le k \le 10$, let $n(k)$ denote the absolute value | (number of 1's in k-th subgroup) minus 4 |. Note $n(k)$ is in the range 0..4, and has the maximum value 4 when either all (eight) of the subgroup's bits are 1's or all are 0's. The fitness of an individual is then defined to be $\Sigma_k(k \cdot n(k))$. The maximum fitness is 1*4 + 2*4 + … + 10*4 = 220. For this problem, there are 2^{10} = 1024 individuals of maximum fitness. These individuals exhibit all 0's or all 1's in each subgroup. As earlier, we used 98% of maximum fitness as acceptable fitness. We ran 20 trials of this experiment, using 40 as our population size, and allowing up to 100 generations per trial. We would expect subgroups to converge towards all 1's or all 0's, with convergence happening sooner in subgroups having a higher group number, and this can be observed when a trial's generations are examined. This time, 13 of the trials found an individual of acceptable fitness before exhausting all 100 generations. Over the 20 trials, the average fitness of the fittest individual found on a trial was 215.5 and the average final generation number was 84.3. See Table 2.

Table 2: Weighted 8-bit Groups

Bits per individual	80
Optimal fitness	220
Acceptable fitness	215.6
Number of trials	20
Population size	40
Max generations	100
Avg final gen. num	84.3
Avg best fitness	215.5

3.3. TARGETING A SPECIFIED INDIVIDUAL

In [Greene, 2001] the following experiment is described. An individual is a 2-dimensional 24 x 24 grid of bits. A particular individual is distinguished (it resembles the letter capital-A against an opposing background); this individual becomes the target. Then, an arbitrary population individual has an *error*, equal to its Hamming distance from the target, with a maximum value of 24 * 24 = 576. Thence the individual has a fitness, defined as maximum error minus own error. Thus maximum fitness also equals 576, and we use 98% of that, or 564.48, as acceptable fitness. We note that this problem is isomorphic to a "counting 1's" problem. In a counting 1's problem, the fitness of an individual is the number of bits having value 1. For the problem at hand, fitness equals the count of bits which have the target's corresponding bit value. In a counting 1's problem, each bit acts as an inde-

pendent building block, there is one individual of maximum fitness, and the fitness landscape everywhere has the same slope upward towards the maximum. In the cited paper, two parents are cut by a random 2-dimensional sub-grid, for purposes of exchanging bit groups at crossover time.

For comparison's sake, first we repeated this experiment. Then secondly we re-worked this problem as a 1-dimensional problem, by re-representing each grid as a 1-dimensional bit string, under row-major representation, and practicing 1-point crossover among parents. Finally, we also used the same problem to test our new linkless self-distancing approach to representation and crossover. We ran 20 trials, allowing up to 2000 generations upon each trial. Table 3 compares the results of these three approaches. For the 1-dimensional and 2-dimensional approaches, all 20 trials exhausted all 2000 generations without unearthing an individual of acceptable fitness. Contrast that with our new approach. It invariably found an individual of acceptable fitness, with average last generation number equal to 496.0. Moreover, this was achieved while using a smaller population size. For this problem, our new approach is unmistakably better and faster.

Table 3: Targeting a Specific Individual

For each representation, an individual is made up of 576 bits, maximum fitness is 576, acceptable fitness is 564.48.

	1-diml	2-diml	LSDGA
Number of trials	20	20	20
Population size	50	50	32
Max generations	2000	2000	2000
Avg final gen. num	2000.0	2000.0	496.0
Avg best fitness	544.05	550.9	565.05

3.4. THE 20 QUEENS PROBLEM

We reprise a second problem from [Greene, 2001]. This is the 20 Queens problem, which is the analogue of the classic 8 Queens problem from chess. There is a 20 x 20 chessboard, and the goal is to have 20 queens placed into board squares so that no queen is attacking the others. In the cited paper, the chessboard surfaces as a 20 x 20 grid of bits, with bit value 1 meaning the board square is occupied by a queen, whereas value 0 means the square is empty. Fitness is addressed as follows. Errors are added up. One or more errors are occurring when any of the following holds: a row or a column contains $n \neq 1$ queens, or a diagonal or counter-diagonal contains $n > 1$ queens. The worst situation occurs when every square (not merely 20 squares) has a queen on it, in which case the total number of errors equals 2*(E - 1)*(2* E - 1), where E = edge-size. Here E = 20 and maximum error = 1482. Then the fitness of an individual is defined as maximum error minus own error. Maximum fitness is then 1482. In our experiments we take 98% of that figure, or 1452.36, to be acceptable fitness.

This is a hard problem, with many constraints to satisfy and many epistatic interactions between bits. The fitness landscape for this problem is hard to analyze, but probably it is rather jagged. Chess players know there are many individuals of maximum fitness. For instance, given one solution to this classic puzzle, the 8 symmetries of a square (obtained by rotations and reflections) provide more solutions.

The cited paper worked this problem 2-dimensionally. In doing so, two parents were cut by a random subgrid, for purposes of crossover. We first re-worked the cited research. Then we also worked this as a 1-dimensional problem, by re-representing a board in row-major form as a 1-dimensional array, and practicing 1-point crossover. Finally we worked this problem using our new approach.

Table 4 summarizes the results. The 1-dimensional and 2-dimensional approaches generally, but not always, found an acceptable individual before exhausting all allowed generations in a given trial. The average best individuals found by the three approaches have nearly equal fitnesses, but our new approach when applied to this problem finds an acceptable individual in *circa* 6 times fewer generations, and does so using a smaller population size, as well. For this problem, our new approach is very much the superior one.

Table 4: The 20 Queens Problem

For each representation, an individual consists of 400 bits, maximum fitness is 1482, acceptable fitness is 1452.36.

	1-diml	2-diml	LSDGA
Number of trials	20	20	20
Population size	100	100	40
Max generations	2000	2000	1000
Avg final gen. num	1863.1	1565.5	286.5
Avg best fitness	1450.6	1451.55	1453.8

3.5. RE: GRAPH BISECTION

The next two experiments concern graph bisection, so in this section we discuss that topic. Let a graph G, having *n* vertices and some number of edges, be given. For simplicity, we will assume *n* is even. A *bisection* of G means a partitioning of G's vertices into two subsets of the same size, *n*/2. The *cut-size* of the bisection is defined to be the number of edges which have an endpoint in each of the two vertex subsets. The *graph bisection problem* is to identify a bisection with the lowest possible cut-size.

Graph bisection has been studied by researchers such as [Kernighan and Lin, 1970], [Johnson *et al*., 1989], and [Laszewski, 1991]. It has also been examined in a series of papers by Bui and Moon and their colleagues, for instance [Bui and Moon, 1993], [Bui and Moon, 1995], and [Bui and Moon, 1996]. Graph bisection has practical applications, and is also known to be a hard problem [Bui and Jones, 1992]. Space does not allow a full explication of the issues of this problem. Our work follows the gen-

eral development of the past researchers. We refer the interested reader to the cited papers.

A bisection of the graph puts its vertices into two "halves", call them the A and B halves. To search for a bisection with minimal cut-size, we proceed as follows. Represent a bisection by using *n* bits, one for each vertex. Bit values of 0 versus 1 signify whether the associated vertex is in the A or B subset. Then we can easily compute cut-size: loop through the list of edges, tallying each whose two endpoints are in different subsets. Note that cut-size can be as low as 1, and can be no greater than the number of edges.

Our attack on a graph bisection problem will use the approach to representation and crossover which we explicated in section 2. An individual in the population consists of *n* bits (*n* = the number of vertices), moreover, we will see to it that *n*/2 of these bits have the value 0 and the other *n*/2 have the value 1. We define the fitness of an individual to be: the number of edges in G, minus the cut-size of the bisection which corresponds to the individual. Maximum fitness equals number of edges, minus 1. When the least possible cut-size of a graph is unknown, as is usually the case, in our trials we let acceptable fitness equal maximum fitness. Mating with crossover can proceed as we described it in section 2, but we need to end it with two additional steps. The first step is a repair step, and the second step is an improvement step.

(There is also another consideration made, at the start of crossover. Our representation admits an isomorphy. The result of flipping every bit of an individual is what we may term its *complement*. An individual and its complement really represent the same partition of the vertex set into two subsets. We note that mating an individual with its (near) complement is likely to produce a chaotically different child, whereas the child of two (near) identical partitions should be a (near) duplicate of its parents. Hence we make it a practice at crossover time that we mate parent-1 with whichever of parent-2 or its complement is the closer to parent-1 in Hamming distance.)

Crossover and mutation can produce an individual (child) for which the A and B subsets are not the same size. This obliges us to repair the individual, by flipping enough of the bits with the value (0 or 1) which occurs in excess.

Our repair work is heuristic. Note that to move a vertex to the other subset (A or B), that is, to flip a single bit, implies a changed bisection and hence a change in cut-size. A negative change to cut-size is favorable, for it makes cut-size become lower, as is our desire. We make a list of the bits which exhibit the excess value (0 or 1), and with each such bit we pair the change in cut-size that would result from flipping it. Repair then takes the form of looping on 2 steps until the required number of bits have flipped: (1) identify the listed bit which has the most favorable change in cut-size, remove it from the list, and flip this bit; (2) update the change-to-cut-size for those listed vertices which are adjacent to the flipped one (this is the only updating which is necessary here, the correct update is to subtract 2).

(The repair work in [Bui & Moon, 1996] is different. Their bits are stored in an array which, for the repair step, is treated as circular. They pick a random index and from there move forward, flipping those bits which exhibit the excess value, until enough have been flipped.)

Next we describe the improvement step. The improvement step, which is done to a partition which has already been balanced into two subsets of the same size, is also heuristic. The idea goes back to [Kernighan & Lin, 1970]. Similar to the earlier observation, we note that to have two vertices, one from each subset, exchange sides also implies a change in cut-size. In the improvement step, a well-chosen subset of the A-elements, together with a well-chosen subset, of the same size, of the B-elements, are identified, then these groups exchange sides. The procedure is summarized in the 5 steps given next. (1) Make a list, *AList*, of the A-elements, pairing each with the change in cut-size that would result if that vertex were to change sides; sort *AList* into increasing order. Similarly form *BList* out of the B-elements. Also create a list *BestPairs*, initially empty.

(2) Now look at a window of the best *w* elements from the *AList* and likewise the *w* best elements from *BList*. Constant *w* is window size; for it we used 10. There are *w***w* pairs of vertices, one from A and one from B, formable from our windows. Identify the pair for which there is the most favorable change to cut-size if the two vertices were to exchange sides; append that pair to the end of list *BestPairs*. (3) Remove these two vertices from *AList* and *BList*. Update the change-to-cut-size of those vertices adjacent to the two vertices. Re-sort *AList* and *BList*. (4) Loop back to step 2 until enough pairs have been put into the sequence *BestPairs*. We follow the advice of Bui and Moon and let *BestPairs* grow to length *n*/6 - 1 where recall *n* is the number of vertices in G. (Kernighan and Lin used the longer length *n* - 1.) (5) Finally, for *k* = 1, 2, 3, etc., consider the partial sums

$$\sum_{i=1}^{k} (\text{change-to-cut-size for i-th pair in BestPairs})$$

The most favorable such partial sum then identifies the *k* elements from A and *k* from B which are made to exchange sides in the improvement step.

3.6. BISECTING THE GRAPH U500.10

The graph named U500.10 is a test case devised by [Johnson *et al*., 1989]. It is a so-called *random geometric graph*, and the authors constructed it as follows. First, 500 points are generated, whose coordinates are random values in the real unit interval [0, 1]. (The 500 points are randomly situated in the unit square.) Then a distance value is calculated, with the property that: when all pairs of points within that distance of one another are connected by an edge, then the expected (that is, average) degree of a vertex is 10.

Bui and Moon have used this same test case. In particular we have in mind [Bui and Moon, 1996]. In that paper the authors provide a detailed comparison of several algorithms, some of their own devising and some from earlier researchers; the algorithms are tested on a multitude of graphs, including U500.10. It is fair to say that the overall best performing algorithm from this paper is the authors' genetic algorithm BFS-GBA (Breadth-First Search Graph-Bisection Algorithm). Another compared algorithm is simulated annealing as practiced in [Johnson *et al*., 1989], denoted SA.

We tested our graph-bisecting form of LSDGA on the graph U500.10. The graph's vertex and edge sets are available on-line at *dimacs.rutgers.edu/pub/dsj/partition.* This graph has 2355 edges, so acceptable fitness is 2354. In Table 5 we summarize how our own algorithm LSDGA measured up against BFS-GBA and SA. To date no one knows the least possible cut-size for U500.10; the least one known is 26. The entries of Table 5 give the least and the average cut-sizes that surfaced over trials. The column for LSDGA comes from our own experiments; the other two columns are copied from [Bui and Moon, 1996]. Our own algorithm found the same least cut-size (26) that the other two algorithms did. Also, on four of our trials our algorithm found a cut-size of 29, which is very close to the least known cut-size. Our algorithm's average cut-size (44.77) falls in between those of the other two algorithms.

Table 5: Bisecting Graph U500.10

Omitted entries in the columns occur when the values are unknown, inappropriate, or are incomparable to LSDGA.

	SA	BFS-GBA	LSDGA
Number of trials	-	-	20
Population size	-	-	32
Max generations	-	-	500
Avg final gen. num	-	-	500
Best cut-size	26	26	26
Avg cut-size	65.8	32.68	44.77

Further comparisons between these three algorithms are difficult to make. The time-costs of SA and BFS-GBA are compared by Bui and Moon, but the costs are given in terms of CPU seconds on particular processors. Algorithm BFS-GBA is a genetic algorithm, but it is a steady-state algorithm whereas ours is generational. Also, the stopping condition used is entirely different; that algorithm stops evolving "when 80% of the population is occupied by solutions with the same quality" (p. 846). Finally, we remain unclear about certain terminology in the paper.

3.7. BISECTING CATERPILLAR GRAPH CAT352

Figure 1 suggests the structure of a so-called *caterpillar graph*. A caterpillar graph is made up of identical star-like clusters, with the star centers connected one to another in a linear sequence. [Bui and Moon, 1996] tell us that caterpillar graphs are especially difficult for certain graph bisection algorithms, such as that of [Kernighan and Lin, 1970], and simulated annealing as practiced in [Johnson *et al*., 1989]. Bui and Moon experimented with several sizes of caterpillar graph. The one named cat352 is made up of stars of 7 vertices, just like in Figure 1. In cat352 there are 50 such stars (giving 350 vertices), plus (we presume, anyway) two terminating vertices of degree 1 at the far left and far right. Plainly the minimal cut-size for graph cat352 is a mere 1, and it corresponds to splitting the graph between the two central stars.

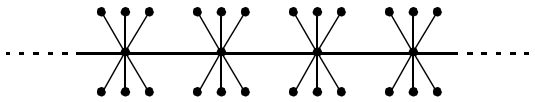


Figure 1: A Caterpillar Graph Segment

We tested our algorithm LSDGA on graph cat352. The number of edges is 351, so our choice of maximum fitness is one less than that, or 350. This is the fitness that is achievable by the optimal bisection. Our stopping condition on each of 20 trials was to either unearth the optimal bisection or stop after 500 generations. On 4 of our 20 trials the optimal bisection was discovered. On the other 16 trials the best bisections we found had cut-sizes of 3 (6 times), 5 (4 times), 7 (3 times), and 9 (3 times). Table 6 summarizes the results, with comparisons to algorithm BFS-GBA of [Bui and Moon, 1996]. The table does not offer a comparison to [Johnson *et al.*, 1989] as they did not use caterpillar graphs. Our best cut-size is the equal of Bui and Moon, though their average cut-size is better than ours.

Table 6: Bisecting Graph cat352

Omitted entries in column two occur when the values are unknown, inappropriate, or are incomparable to LSDGA.

	BFS-GBA	LSDGA
Number of trials	-	20
Population size	-	32
Max generations	-	500
Avg final gen. num	-	431.95
Best cut-size	1	1
Avg cut-size	2.25	4.5

4. CONCLUSIONS

We have presented a new representation and crossover algorithm for genetic algorithms. Bits are not linked together at all. On each generation, the current population is used to calculate a pseudo-distance between bits. Bits which are close under the pseudo-distance are ones which appear to belong to a same building block. Under crossover, nearby bits get copied in clumps into the children. Thus our new linkless self-distancing approach does

directly what other approaches to learning linkage and preserving building blocks have only hoped to do indirectly. Experiments were performed on six problems. These included simple problems, problems with numerous equally fit optima, and very challenging problems in graph bisection. Our new approach worked very successfully on these problems, equaling and often outperforming other more familiar approaches.

5. FUTURE WORK

Our approach is brand new. There are many questions that suggest themselves, which we have not yet had time to pursue. Are there better choices for our system parameters than the ones we have cited? Is there a better measure of fitness concentration than the entropy calculation we have used? Is there a Schema Theorem for our approach? In what ways can premature convergence occur, and how can it be combatted? Can careful bookkeeping allow us to adapt the approach to steady-state GA's? The approach needs to be extended to the case that the granularity of a gene is bigger than a single bit. In that vein, there would be many more than 4 ways that a pair of genes can assume a pair of values. Will our entropy calculation for fitness concentration, properly adapted, still succeed? In short, there are many issues inviting exploration.

References

Bui, T. N., and C. Jones (1992). "Finding Good Approximate Vertex and Edge Partitions is NP-Hard," *Information Processing Letters*, vol. 42, pp. 153-159.

Bui, T, N., and Moon, B.-R. (1993). "Hyperplane Synthesis for Genetic Algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 102-109. Morgan Kaufmann Publishing, San Mateo, CA.

Bui, T. N., and B.-R. Moon (1995). "On Multi-Dimensional Encoding/Crossover," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 49-56. Morgan Kaufmann Publishing, San Francisco, CA.

Bui, T. N., and B.-R. Moon (1996). "Genetic Algorithm and Graph Partitioning", *IEEE Transaction on Computers*, vol. 45, no. 7, pp. 841-855.

Cohoon, J. P., W. N. Martin, and D. S. Richards (1991). "A Multi-Population Genetic Algorithm for Solving the k-Partition Problem on Hypercubes," *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 244-248.

Goldberg, D., and Lingle, R. (1985). "Alleles, loci, and the traveling salesman problem", in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pp. 154-159.

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

Goldberg, D. E., Korb, B., and Deb, K. (1989). "Messy Genetic Algorithms: Motivation, Analysis, and First Results," in *Complex Systems*, vol. 3, pp. 493-530.

Greene, W. A. (2000). "A Non-Linear Schema Theorem for Genetic Algorithms," in the *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2000), pp. 189-194.

Greene, W. A. (2001). "Non-Linear Bit Arrangements in genetic Algorithms," in *2001 Genetic and Evolutionary Computation Conference Late-Breaking Papers*, pp. 138-144.

Harik, G. (1999). *Linkage Learning via Probabilistic Modeling in the ECGA*. IlliGAL Report No. 99010; Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana. IL.

Holland, John (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Johnson, D. S., C. Aragon, L. McGeoch, and C. Schevon (1989). "Optimization by Simulated Annealing: an Experimental Evaluation, Part 1: Graph Partitioning", *Operations Research*, vol. 37, pp. 865-892.

Kernighan, B., and S. Lin (1970). "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technical Journal*, vol. 49, pp. 291-307.

Laszewski, G. (1991). "Intelligent Structural Operators for the k-Way Graph Partitioning Problem," *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 45-52.

Muehlenbein, H., and Paass, G. (1996). "From Recombination of Genes to the Estimation of Distributions, I: Binary Parameters," in *Parallel Problem Solving from Nature IV*, pp. 178-187. Springer-Verlag, Berlin.

Pelikan, M., Goldberg, D. E., and Cantu-Pas, E. (1998). *Linkage Problem, Distribution Estimation, and Bayesian Networks*. IlliGAL Report No. 98013; Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana. IL.

Sehitoglu, O. T., and G. Ucoluk (2001). "A Building Block Favoring Reordering Method for Gene Positions in Genetic Algorithms," in the *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2001), pp. 571-575.

Syswerda, G. (1989). "Uniform Crossover in Genetic Algorithms", in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 2-9. Morgan Kaufmann Publishing, San Mateo, CA.

Exploring the Parameter Space of a Genetic Algorithm for Training an Analog Neural Network

Steffen G. Hohmann, Johannes Schemmel, Felix Schürmann, Karlheinz Meier
Kirchoff-Institute for Physics, Heidelberg University, Schröderstrasse 90,
69120 Heidelberg, Germany
e-mail: hohmann@kip.uni-heidelberg.de
phone: Germany-(0)6221-544329

Abstract

This paper presents experimental results obtained during the training of an analog hardware neural network. A simple genetic algorithm is used to optimize the synaptic weights. The parameter space of this algorithm has been intensively scanned for two learning tasks (4 and 5 bit parity). The results provide a quantitative insight into the interdependencies of the evolution parameters and how the optimal settings are predetermined by the learning problem. It is observed that population sizes in the order of 15 in connection with mutation rates of about 1% yield the best performance of the training when using moderate selection. The optimal population size is found to be independent of the learning task. A significant improvement of the training success can be achieved, when the role of crossover is reduced and higher mutation rates combined with stronger selection are applied. The observations are shown to be essentially independent of the signal to noise ratio within the analog hardware.

1 Introduction

Artificial neural networks as a computational model and simulated evolution as an optimization procedure are both biologically inspired concepts that have been successfully applied to numerous problems. Their combination to evolutionary artificial neural networks (EANNs) has been widely studied in the past years [Yao, 1999]. The processing of information within a neural network allows a high degree of parallelism, which motivates a special hardware implementation. Our group has recently developed an analog neural network chip optimized for the training with evolutionary algorithms that exploits the high degree of parallelization possible in modern VLSI[1] technologies [Schemmel, 2001]. The chip is integrated in a training and test setup that allows processing of in the order of 1000 individuals per second. This makes it feasible to approach another critical aspect of evolutionary optimization: The performance of an evolutionary algorithm depends on the tuning of several parameters that can have a massive impact on the success and the speed of the optimization process [De Jong, 1975]. Depending on the applied algorithm and the chosen problem, the interrelation of the various parameters can be very complex. Although recent theoretical investigations on the dynamics of genetic algorithms show promising results [Rogers, 2001], the complexity of realistic problems - e.g. the training of neural networks - greatly obstructs their theoretical analysis. In practice the optimal evolution parameters for a given task and a specific algorithm still have to be approximated by trial and error, guessed by experience or have to be found by other optimization procedures themselves [Grefenstette, 1986]. What additionally impedes thorough investigations of the interaction of evolution parameters is the limited amount of available experimental data, as it is usually costly to obtain [Schaffer, 1989], [De Jong, 1975].

This paper discusses the results obtained from more than 260,000 evolution runs carried out to train neural networks for two different tasks, which corresponds to several billions of tested individuals. A total of about 950 parameter settings are evaluated and analyzed with respect to the success of the learning process within a fixed number of fitness evaluations. The aim of this investigation is to clarify the influence of the various evolution parameters on the performance of the training algorithm. To promote a deeper understanding of the results and to motivate further theore-

[1] Very Large-Scale Integration

tical analysis, the genetic representation and the evolutionary algorithm have been chosen as to maintain a close resemblance to classical genetic algorithms (GAs) [Goldberg, 1989]. Furthermore, the tackled learning tasks (4 and 5 bit parity) are small in size, but known to be challenging problems for neural network training. It is assumed that the main aspects of the results can qualitatively be transfered to more complex tasks.

2 Hardware Realization of the Neural Network

The analog neural network chip used throughout the experiments is based on the concept of a feedforward network. It consists of 64 × 64 synapses that connect each of the 64 input neurons with each of the 64 output neurons. The analog operation of the chip is limited to the synaptic weights and the inputs of the output neurons. Both, input (I_j) and output signals (O_i) of the network are digital. A synapse is activated, when the value of the corresponding input neuron is set to 1. This way, the weight multiplication is reduced to an addition: The weight of each synapse (w_{ij}) is stored as charge on a storage capacitor and each output neuron compares the sum of the signals of all its activated synapses with a fixed reference voltage (b). If the reference value is exceeded, the neuron fires.

Throughout the presented experiments, the chip is operated in a special mode: All input neurons are combined to pairs. Each pair is used as one differential input. This is done automatically by setting the value of every second input neuron to the inverted value of its neighbour: If one input neuron is switched off, its inverted counterpart is switched on and vice versa. In this *combine* mode the operation of the network can be modeled by a slightly modified Perceptron formula with the activation function $g(x)$ of the output neurons set to the Heaviside function $\Theta(x - b)$:

$$O_i = \Theta\left(\sum_{j=0}^{31}[w_{i,2j+1} \cdot I_{2j+1} + w_{i,2j+2} \cdot (1 - I_{2j+1})] - b\right)$$
$$\text{with } I, O \in \{0, 1\} \tag{1}$$

Hence, the connection of one combined input neuron to one specific output neuron is characterized by a set of two weights. The effective number of available input neurons is reduced to 32. On the other hand, the number of activated input neurons remains the same, independent of the input data. This has several advantages for the performance of the chip. A further discussion of these topics lies beyond the scope of this article. For more detailed technical information see [Schemmel, 2001].

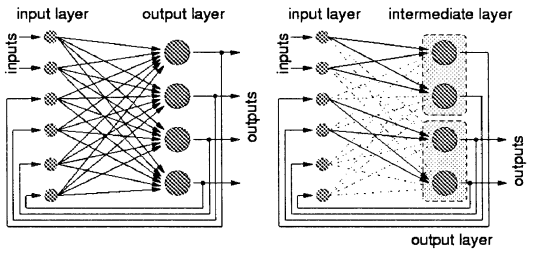


Figure 1: **Left**: A recurrent network. **Right**: Configured as a two-layer network by setting some synapses to zero (dashed lines).

The network operates within a discrete time update scheme, i.e. Equation 1 is calculated once for each network cycle. For the chip to be used as a recurrent network, the signals of half of the output neurons can be fed back to the input neurons. In this mode the output of the network at a given time does not only depend on the actual input, but also on the previous network cycle. The feedback can be used to configure the network as a virtual multi-layer Perceptron if the appropriate weights are set to zero. Both cases are schematically illustrated in Figure 1. If the chip is used as a multi-layer network with n layers, a corresponding number of network cycles is used to propagate a signal from the input to the output neurons. The maximum possible network frequency exceeds 50 MHz, which results in more than 200 giga connections per second and makes the implementation of networks with several layers practical.

3 Implementation of the Training Algorithm

The network is trained by a generational genetic algorithm with ranking selection, a chromosomewise one-point crossover and a single-gene mutation operator. Throughout our experiments the architecture of the network - i.e. the number of used input and output neurons, the active feedback connections and the number of network cycles - is fixed for each learning problem. The genome only contains information about the synaptic weights of the individual networks to configure the chip.

3.1 Genetic Representation

A genome consists of 64 chromosomes maximum, each describing the complete set of 64 synaptic weights that connect one output neuron to all input neurons. One weight is coded as a floating point number from the interval [−1, 1] and is regarded as a complete single

gene. Within one chromosome these weights are arranged to form a linear string.

Neither all neurons, nor all synapses of the array are necessarily used for a learning problem. In practice the genome only contains those chromosomes which parameterize the output neurons of the chip that are used either as inner neurons in hidden layers or as actual output neurons of the network. For technical reasons the chromosomes still contain information about all connecting synapses even if not all of the 64 input neurons are used. The parts of the genome which describe unused synapses have negligible effect on the performance of the represented network and thus do not influence the fitness of the individual.

3.2 Genetic Operators

During recombination a common one-point crossover operator is used and is successively applied to each corresponding pair of chromosomes provided by the two involved individuals. A new cut point is randomly and uniformly selected for each chromosome pair. The weights are regarded as the smallest units of the genome and always swapped as a whole. Accordingly, the mutation operator acts on the complete gene by replacing the weight to be mutated by a new value randomly and uniformly selected from the interval $[-1, 1]$. Both, crossover and mutation do not distinguish between parts of the genome that are relevant for the fitness and those which are inactive.

3.3 Selection and Reproduction Scheme

The algorithm acts on a population of ϕ genomes and the initial population is chosen by setting all genes of all individuals to random values uniformly distributed within the interval $[-1, 1]$. After evaluation of the fitness of each individual, the new generation is created from the preceeding population by successively executing three steps: selection, recombination and mutation. The selection scheme is rank based and is controlled by a parameter ρ called the replace percentage. After a fitness ranking of all individuals, the worst $n = \lfloor \rho \cdot \phi \rfloor$ genotypes are replaced by the best individual of the population. As long as $\rho \neq 0$ the worst individual is replaced even if $n \leq 1$. Thus if γ_i denotes the genotype at position i in the ranking we can write:

$$\gamma_i = \gamma_1 \quad \forall i \in \begin{cases} \{\phi - n + 1, \ldots, \phi\} & \text{if } n \geq 1 \\ \{\phi\} & \text{if } n < 1 \wedge \rho \neq 0 \\ \{\} & \text{if } \rho = 0 \end{cases}$$
$$n = \lfloor \rho \cdot \phi \rfloor \qquad (2)$$

In the following step, all n replaced individuals are consecutively crossed with a new partner randomly selected among the whole population. Since γ_1 and the last n genotypes of the population are identical, it is possible that an individual is actually combined with a clone of itself which renders the crossover operator noneffective. On the other hand, one individual can be chosen more than once for crossover and may be changed repeatedly with cumulating effects. After the recombination step, the whole population is mutated whereby every single gene of each genotype in the population is changed with a fixed probability μ in the way described in 3.2. The resulting population of genotypes after the mutation step forms the new generation.

It should be noted that due to the applied mutation strategy even the best individual is not guaranteed to survive the generation step completely unchanged. Hence, the algorithm does not implement an elitist reproduction scheme in the sense of the usual definition [De Jong, 1975] though it is clearly biased towards a fast reproduction of the best indivdual.

3.4 General Considerations

In respect to the applicability of training neural networks, more sophisticated algorithms or codings are conceivable and have been numerously presented in the literature [Yao, 1999]. On the other hand, both, the presented coding and the algorithm combine direct compatibility to the used hardware with high simplicity, allow fast implementation and have proven to be capable of successfully training a neural network. Furthermore, the genetic representation of the problem and the applied operators retain a close resemblance to those used in classical GAs [Goldberg, 1989]. On the genotype level, the whole algorithm can completely be described by three variables ϕ, ρ and μ. This should allow for a better understanding of the performance of this algorithm depending on the parameters.

4 Experimental Setup

4.1 Technical Environment and Scale Factor

The network chip is connected to a standard PC by a PCI interface board that carries its own RAM and a field programmable gate array (FPGA). In the current stage of the setup, the training algorithm is implemented in C++ and executed on the host computer. At the beginning of an evolution run the testpatterns and target data of the learning task are sent to the RAM of the PCI-card where they remain throughout

learning problem	4BP	5BP
used genes	38	58
total genes	256	320
ineffective genes	218	262
No. of testpatterns	16	32
maximum fitness	160	320
threshold	157	317
individuals/sec	1061	949

Table 1: Specification of the applied learning problems.

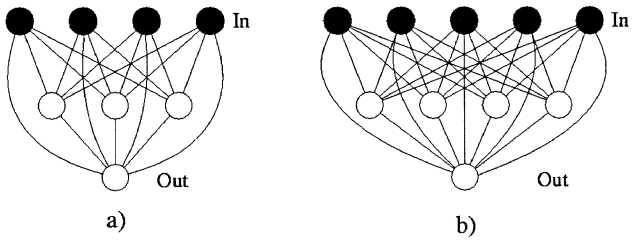


Figure 2: The used architectures for the 4BP problem (a) and the 5BP problem (b).

the whole evolutionary process. During the evolution the data for each individual is sent to the network via the FPGA using a 16 bit digital to analog converter (DAC) that generates the analog weight values from the gene data. Once the weight values have been stored within the synapse array, the testpatterns are consecutively applied to the network. The results are read back by the FPGA and are used to calculate the fitness of the individual.

Due to the analog operation of the chip the reference voltage and weight values in the network are subject to fluctuations. The repeated application of the same input pattern does not necessarily yield identical outputs, especially, if the input signals of one or more output neurons are close to the reference value. For the selected problems it is thus necessary to present each test pattern repeatedly. The performance of the network can be evaluated by counting the number of correct bits in the output of the network summed over all successive presentations of the pattern. Besides reflecting the stability of the solution, this number also yields a smoother performance measure. It is therefore used as the fitness of the individual.

The range of possible weight values to be stored as charge within the synapse array is technically limited. The gene values from the interval $[-1, 1]$ have to be mapped onto this range of possible weights. We thus introduce the weight scaling factor $\omega \leq 1$ that controls this mapping. If $\omega = 1$, the whole technically possible range is exploited. If $\omega < 1$, only the corresponding fraction of the full range is used. Since the absolute average size of the signal fluctuations remains constant independent of the choice of ω, the variation of this parameter can be used to control the signal to noise ratio of the weights. The latter is expected to have considerable impact on the performance of the network.

4.2 Learning Task, Network Architecture and Success Condition

Within the presented experiments the network is trained for two tasks: 4 bit parity (4BP) and 5 bit parity (5BP). Being a linearly nonseparable problem the calculation of parity requires the chip to be configured as a two layer network [Hertz, 1991]. Hence, a number of two network cycles is used in each case. The hidden layer consists of three and four neurons for the 4BP and 5BP problems respectively. Since no weight is forced to be zero, shortcut connections from the input layer directly to the output neuron are possible. Figure 2 shows the resulting architectures for both problems. Similar architectures for the N bit parity task are reported in the literature [Pujol, 1998]. Since the neurons are operated in *combine* mode (Section 2), all synaptic connections are actually described by two separate weights that have to be optimized independently. For 4BP this results in $19 \times 2 = 38$ used weight values (see Figure 2). As the genome describes the complete set of synaptic weights for each of the four used output neurons (Section 3.1), it contains $4 \times 64 = 256$ genes, leaving 218 genes ineffective. The corresponding values for 5BP are shown in Table 1. To take into account the effects introduced by noise, each test pattern of the current task is applied ten times throughout all experiments which multiplies the maximum possible fitness by ten. The training is considered successful if the fitness of the best individual in the population averaged over the last five generations is better than a given threshold. The resulting values of maximum possible fitness and the chosen threshold values can also be found in Table 1. The thresholds are chosen empirically as to allow for small fluctuations in the performance of the found solution, which is a realistical approach for the setup used. Due to the applied success criteria, the length of an evolution run can obviously not be measured with a higher precision than ± 5 generations. The approximate average number of individuals per second that can be processed by the setup depends on the population size ϕ and the number of testpatterns that are applied for the fitness evaluation. The corresponding values, when using a standard PC[2] and a population size of 100, are included in Table 1.

[2]In this case an AMD Athlon XP 1700 is used.

5 Experiments and Results

Several measures for the performance of a genetic algorithm are conceivable [De Jong, 1975]. Since it is a stochastic search procedure, comparing the performance among variations of a genetic algorithm with different parameters is not unproblematic [Schaffer, 1989]. The probability of the algorithm to reach the success condition described in 4.2 within a given number of fitness evaluations depends on the chosen evolution parameters. It will be regarded as the performance measure of a parameter configuration throughout the remainder of this article and is referred to as the yield Y given in percent. This rather practical approach has been chosen because the number of fitness evaluations is nearly proportional to the computation time needed by the serial implementation of the algorithm.

As discussed in the previous sections, the used training algorithm can be controlled by four parameters: The population size ϕ, the replace percentage ρ and the mutation rate μ (Section 3) parameterize the simulated evolution, while the weight scaling factor ω is specific for the used analog hardware (Section 4.1). The goal of the conducted experiments is to illuminate the influence of these parameters on the yield Y. Given a defined parameter setting and a learning task the yield is evaluated by conducting a number of N_E evolution runs and counting the successes N_S to calculate the success probability p and the yield using $p = N_S/N_E = Y/100$. Since all evolution runs are independent, we can assume the outcome of the experiment to be binomially distributed and estimate the error of the measured Y as $\Delta Y = \sqrt{pq/N_E} \times 100$ with $q = 1 - p$. The measurand Y is evaluated for parameter settings located on three selected two-dimensional hyperplanes of the four-dimensional parameter space as a complete scan would be impracticable. The following sections summarize the results obtained during the scans of the investigated subspaces for the various learning problems. The experiments have been conducted using two identical setups that were continuously running for a period of about eight weeks.

5.1 Population Size and Mutation Rate

This section presents the results of a simultaneous variation of the parameters ϕ and μ while keeping $\rho = 20\%$ and $\omega = 0.9$. For each learning task ϕ and μ have been varied within the ranges shown in Table 2, which also lists the total number of parameter settings n_s that has been evaluated for each problem. To use the computation time more efficiently, the trials have not been distributed uniformly within these ranges but have been clustered in regions with stronger variations of $Y(\phi, \mu)$. The results are qualitatively the same for both tasks and their analysis shall be discussed for 4BP as an example. Figure 3 shows the yield Y as a function of the mutaion rate μ for three different values of the population size ϕ. Given a fixed ϕ the yield shows a clear dependence of μ: Y exhibits a maximum at a specific ϕ-dependent value $\mu_{max}(\phi)$. While $\mu_{max}(\phi)$ itself increases with growing ϕ, the maximum yield $Y(\mu_{max})$ decreases. A closer examination reveals that for a fixed ϕ the data can be fitted satisfactorily by the function

$$Y_F(\phi, \mu) = F_1(\phi) \cdot \mu^{F_2(\phi)} \cdot e^{F_3(\phi) \cdot \mu^2} \qquad (3)$$

and that the ϕ-dependece of the data can be modeled by setting

$$F_i(\phi) = f_{i1} \cdot \phi^{f_{i2}} + f_{i3} \quad , \quad i \in \{1, 2, 3\}, \qquad (4)$$

whereby we introduce the nine fit parameters f_{ij}. The combination of Equations 3 and 4 is fitted to the whole set of n_s evaluated points $Y_{kl} = Y(\phi_k, \mu_l)$, i.e. the Gauss-Newton method[3] is used to set the fit parameters with regard to the minimization of

$$\chi^2 = \sum_{k,l} \frac{(Y_{kl} - Y_F(\phi_k, \mu_l))^2}{\Delta Y_{kl}^2} \quad . \qquad (5)$$

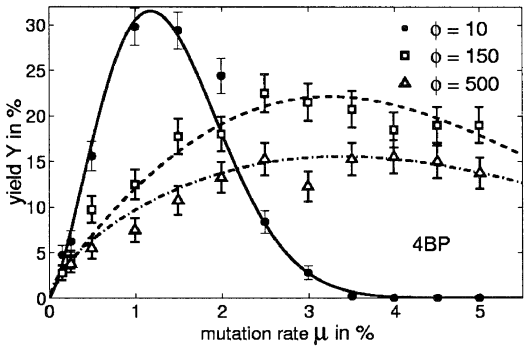


Figure 3: The yield Y as a function of μ for different values of ϕ and the 4BP problem ($\rho = 20\%$).

learning problem	4BP	5BP
population size ϕ	5-700	5-600
mutation rate μ in %	0.15-5	0.15-5
max. No. fitness eval.	16800	39600
No. of tested settings (n_s)	239	153
N_E per setting	400	300
opt. pop. size (ϕ_{opt})	14	13
opt. mut. rate (μ_{opt})	1.38	0.83
max. yield (Y_{opt}) in %	31.8	34.0
$\chi^2/n_{d.o.f.}$	1.36	2.96

Table 2: Parameters and results of the ϕ vs. μ experiments.

[3]The fit routine is implemented in MATLAB.

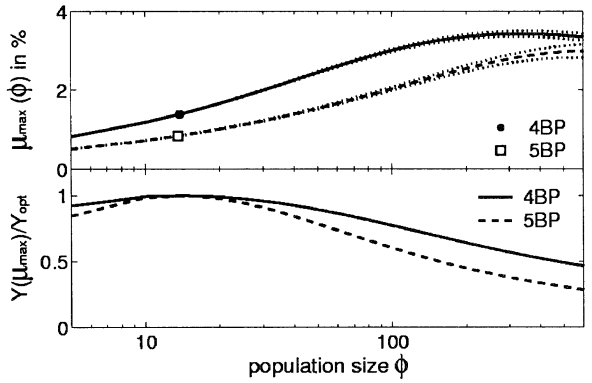


Figure 4: Fit results: The optimal mutation rate $\mu_{max}(\phi)$ and the normalized yield $Y(\mu_{max})/Y_{opt}$ as a function of ϕ for both tasks ($\rho = 20\%$). The symbols mark the parameters for overall maximum yield.

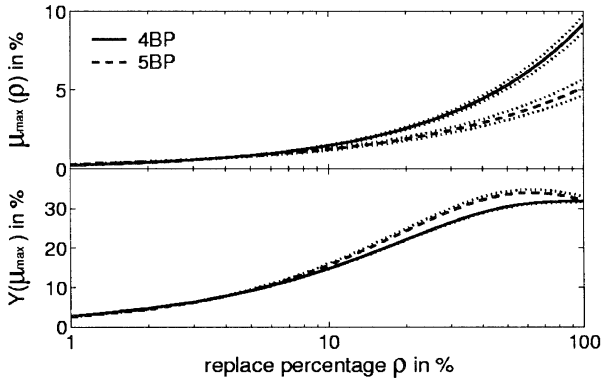


Figure 5: Fit results: $\mu_{max}(\rho)$ and $Y(\mu_{max})$ as a function of ρ for both tasks ($\phi = 100$).

The fit results for the shown values of ϕ are presented in Figure 3 as line plots. It can be seen that the fit reproduces the data. The resulting χ^2 divided by the number of degrees of freedom $n_{d.o.f.} = n_s - 9$ is given in Table 2. The same fit procedure can be executed for 5BP, yielding different sets of f_{ij}. It is to be noted that Equations 3 and 4 are chosen empirically and are not motivated by a theoretical model so far. Nevertheless, they describe $Y(\phi, \mu)$ for both learning tasks reasonably well as can be seen from the values $\chi^2/n_{d.o.f.}$ in Table 2. For the following analysis the maximum number of fitness evaluations allowed for an evolution to fulfill the success condition is set for each learning task separately (see Table 2). The values are chosen as to lead to comparable maximum yields when using the respective optimal evolution parameters predicted by the fits. This is done to assure that the results of the analysis allow a comparison between the tasks although they vary in difficulty.

Given the two sets of fit parameters f_{ij} that describe the landscapes $Y(\phi, \mu)$ for each task, one can study the resulting function $\mu_{max}(\phi)$, which is shown in the upper half of Figure 4 within the investigated ranges on a logarithmic ϕ-scale. The calculated parameters f_{ij} exhibit a certain error due to the uncertainty ΔY of the measured Y. This leads to a finite precison of the prediction for $\mu_{max}(\phi)$ illustrated by the dotted lines. Both curves exhibit a similar behaviour: The optimal mutation rate shows a strong increase with ϕ for small population sizes but saturates for large populations. For 4BP it even slightly decreases for higher values of ϕ. Investigations on other optimization problems find an optimal mutation rate that is independent of the population size [Rogers, 2001] while others report the opposite correlation [Schaffer, 1989]. The lower half of Figure 4 displays the yield $Y(\mu_{max})$ at the optimal mutation rate for a given ϕ divided by the yield Y_{opt} reached in the global maximum (ϕ_{opt}, μ_{opt}). It is remarkable that the yield displays a distinct maximum at a population size that is nearly equal for both tasks. Comparison to the upper half of Figure 4 reveals, that the regions of roughly constant μ_{max} are in fact regions of a low yield compared to the global maximum. The parameters (ϕ_{opt}, μ_{opt}) that lead to the highest possible yield as predicted by the fits are shown in Table 2 and are marked in Figure 4. Besides the similarity in the respective values of ϕ_{opt} it is noticable that they are significantly lower than those commonly suggested ([DeJong, 1975], [Grefenstette, 1986]) but are higher than the lowest possible ϕ as recommended by theoretical investigations concerning serial genetic algorithms [Goldberg, 1989b]. Other investigations on function optimization find optimal mutation rates that depend on the length l of the bit string used as genome like $\mu_{opt} = 1/l$ [Bäck, 1993]. The values in Table 2 seem to be in accordance with this observation if l is defined as the number of used genes (Table 1). However, since the used coding differs from a binary representation, the application of this formula is problematic.

5.2 Replace Percentage and Mutation Rate

In a set of experiments similar to those in Section 5.1 the replace percentage ρ and mutation rate μ have been varied while keeping $\phi = 100$ and $\omega = 0.9$. The scanned ranges of ρ and μ and the total number of evaluated settings is listed in Table 3. Once again the results are qualitatively the same for both tasks and the analysis of the data proceeds similar to Section 5.1: For a fixed replace percentage ρ the dependency of the yield $Y(\rho, \mu)$ on μ can be fitted by the function

$$Y_F(\rho, \mu) = G_1(\rho) \cdot \mu^{G_2(\rho)} \cdot e^{G_3(\rho) \cdot \mu^2} \qquad (6)$$

learning problem	4BP	5BP
replace percentage ρ in %	5-100	10-100
mutation rate μ in %	0.5-10	1-10
max. No. fitness eval.	16300	40000
No. of tested settings (n_S)	270	62
N_E per setting	400	300
opt. rep. pctg. (ρ_{opt}) in %	100	65
opt. mut. rate (μ_{opt})	9.2	4.0
max. yield (Y_{opt}) in %	32.0	34.5
$\chi^2/n_{d.o.f.}$	1.37	3.31

Table 3: Parameters and results of the ρ vs. μ experiments.

and a closer investigation reveals, that the ρ-dependency can be modeled as

$$G_1(\rho) = g_1 \cdot e^{g_2 \cdot \rho} + g_3 \quad (7)$$

$$G_2(\rho) = g_4 \quad (8)$$

$$G_3(\rho) = g_5 \cdot \rho^{g_6} \quad . \quad (9)$$

Note that the exponent $G_2 = g_4$ of μ in Equation 6 does not depend on ρ and that this model uses just six parameters instead of nine as in Section 4. Fitting the set of Equations 6-9 to the data, one finds that $g_4 = 1.008$ for 4BP and $g_4 = 1.514$ for 5BP. If g_4 is not regarded as a fit parameter but is set manually to 1 and 1.5 for 4BP and 5BP respectively, the quality of the fit does not suffer measurably. The resulting values of $\chi^2/n_{d.o.f.}$ are shown in Table 3. It is remarkable that g_4 apparently exhibits constant simple values that seem to depend only on the learning task. While this article is written, additional data is gathered for the 5BP task. More data points will improve the quality of the fit and might allow a more reliable prediction for g_4. However, it can be foreseen that the conclusions of the following analysis will not be affected.

Given the fit parameters g_l, it is possible to investigate the optimal mutation rate μ_{max} for a given ρ and the yield $Y(\mu_{max})$ at this point. Figure 5 shows the results. For both tasks, the optimal mutation rate μ_{max} increases monotonically with increasing ρ. This is understandable, since the diversity in the population is reduced significantly when ρ increases. Thus the introduction of new genes by mutation becomes more important. The yield $Y(\mu_{max})$ also increases with ρ. For 4BP the maximum possible yield is in fact obtained at $\rho = 100\%$, while for 5BP a maximum is reached at $\rho \approx 65\%$. Additional data will clarify if the latter observation is an authentic feature or an effect introduced by the limited precision of the fit. However, it is obvious that high replace percentages ρ in combination with high mutation rates μ result in an optimal yield Y. Note that the choice of $\rho = 100\%$ renders the crossover operator noneffective and the search is driven solely by the selection of the best individual and the parallel mutation of its ϕ clones. It has previously been suggested, that this kind of strategy leads to better performance than algorithms based mainly on recombination [Schaffer, 1989]. Especially, with regard to the training of neural networks, the crossover operator is known to be problematic [Yao, 1999]. The presented results verify these considerations.

learning problem	4BP	5BP
scale factor ω	0.13-1.0	0.13-1.0
mutation rate μ in %	0.5-4.5	0.6-3.6
max. No. fitness eval.	20000	40000
No. of tested settings (n_S)	126	84
N_E per setting	400	300
opt. mut. rate (μ_{opt}) in %	2.6	1.6

Table 4: Parameters and results of the ω vs. μ experiments.

5.3 Weight Scaling and Mutation Rate

In this section we discuss the effects of a variation of the weight scaling factor ω on the success of the training algorithm. At $\phi = 50$ and $\rho = 20$ the yield Y is measured for various settings of μ and ω as given in Table 4. In accordance to the previous observations, Y reaches a maximum at a certain optimal mutation rate μ_{max} for each value of ω. The maximum can be found by fitting the data with a function similar to those presented in the foregoing sections:

$$Y_F(\omega, \mu) = H_1(\omega) \cdot \mu^{H_2(\omega)} \cdot e^{H_3(\omega) \cdot \mu^2} \quad (10)$$

This is done for each value of ω. The determined optimal mutation rates $\mu_{max}(\omega)$ together with the corresponding yields $Y(\omega, \mu_{max})$ are presented in Figure 6. It can be seen that the results are essentially independent of the learning task: Small values of ω result in a low yield or even inhibit the success of the training completely. Using a small scale factor leads to a poor signal to noise ratio of the weights and the performance of the network suffers accordingly. Under these condidtions it is not possible for the algorithm to find a stable solution. An increase of ω improves the signal to noise ratio. This makes it easier for the algorithm to find a network with a stable performance despite the presence of noise. Thus the maximum yield increases. For a scale factor higher than approximately 0.6 the yield is seen to be independent of ω within the given precision. It can be concluded that the success of the training is not limited by the noise of the weights once a specific signal to noise ratio is exceeded.

The errors of the derived optimal mutation rates depend on the distinctness of the respective maximum in $Y(\omega, \mu)$ and on the precision of the data. The latter decreases, when the yield is small. The clearness

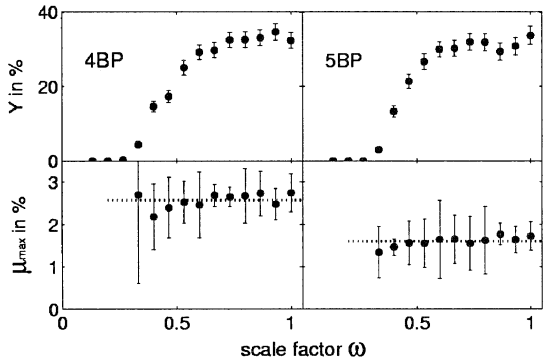


Figure 6: Yield Y and optimal mutation rate μ_{max} as a function of ω for 4BP (left) and 5BP (right).

of the maximum varies slightly between the measurements at different ω. Within the estimated error the optimal mutation rate μ_{max} shows no dependency on the scale factor ω and the signal to noise ratio. This even holds for regions of ω, where the yield Y is still seen to be affected by the weight noise. The previously described experiments were conducted using $\omega = 0.9$. In this domain, both, the optimal mutation rates and the yields are not measurably influenced by the weight fluctuations. It can be inferred that the presented results concerning the interaction of the parameters are not influenced by the presence of noise. However, it shall be repeated that the weights are still subject to fluctuations even for $\omega = 1$. It is not possible to test the system without any noise. On the other hand, software simulations of the system indicate that a minimal level of noise can in fact be advantageous for the training. Future studies with a new generation of analog network chips [Schemmel, 2002] have to investigate the behaviour of the system under a further reduction of the signal noise.

6 Conclusions

We have presented detailed experimental data showing the influence of the evolution parameters on the success of the training of an analog neural network. A quantitative analysis of the data has been conducted and reveals strong dependencies between the parameters. The results suggest that population sizes in the order of 15 and mutation rates of about 1% are to be preferred if the algorithm relies mainly on recombination and applies moderate selection. The optimal population size seems to be independent of the learning problem. The performace of the algorithm improves significantly when a stronger selection is applied and recombination is given only a small role or is even neglected completely. In this case, emphasis has to be placed on mutation, thus higher mutation rates are favorable. We have shown, that these conclusions are not influenced by the noise of the analog weight values within the chip and should in principle be transferable to other systems. At this moment, we do not have a thorough theoretical understanding of the results. It is reasonable to assume that the qualitative behaviour common to both tasks might be a feature of the algorithm, while the concrete values of the optimal parameters could in principle be connected to characteristics of the specific tasks and the used network architectures. We hope that our results encourage further theoretical or experimental research in this direction[4].

References

T. Bäck (1993). Optimal Mutation Rates in Genetic Search. *International Conference on Genetic Algorithms*, University of Illinois, Urbana-Champaign, pp. 2-8.

K. DeJong (1975). The Analysis and Behaviour of a Class of Genetic Adaptive Systems. *PhD thesis*, University of Michigan. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76-9381.

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

D. E. Goldberg (1989b). Sizing Populations for Serial and Parallel Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA, pp. 70-79.

J. J. Grefenstette (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16, No. 1, pp. 122-128.

J. A. Hertz, A. Krogh, R. G. Palmer (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, Inc., Redwood City, CA.

J. Pujol, R. Poli (1998). Evolving Neural Networks Using a Dual Representation with a Combined Crossover Operator. *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC)*, pp. 416-421.

A. Rogers, A. Prügel-Bennett (2001). A Solvable Model of a Hard Optimization Problem. *Theoretical Aspects of Evolutionary Computing*, pp. 209-224, Springer.

J. D. Schaffer, R. A. Caruana, L. J. Eshelman, R. Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA, pp. 51-60.

J. Schemmel, K. Meier, F. Schürmann (2001). A VLSI Implementation of an Analog Neural Network suited for Genetic Algorithms. *Proceedings of the International Conference on Evolvable Systems 2001*, Springer, pp. 50-61.

J. Schemmel, F. Schürmann, S. Hohmann, K. Meier (2002). An Integrated Mixed-Mode Neural Network Architecture for Megasynapse ANNs. To appear in the *Proceedings of the International Joint Conference on Neural Networks 2002.*

X. Yao (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, Vol. 87, No. 9, pp. 1423-47.

[4] The data and the MATLAB macros are available at *http://www.uni-heidelberg.de/vision/projects/evo_fpnn_related.html* .

A Permutation Genetic Algorithm for Variable Ordering in Learning Bayesian Networks from Data

William H. Hsu **Haipeng Guo** **Benjamin B. Perry** **Julie A. Stilson**

Laboratory for Knowledge Discovery in Databases, Kansas State University
234 Nichols Hall, Manhattan, KS 66506
{bhsu | hpguo | bbp9857 | jas3466}@cis.ksu.edu http://www.kddresearch.org

Abstract

Greedy score-based algorithms for learning the structure of Bayesian networks may produce very different models depending on the order in which variables are scored. These models often vary significantly in quality when applied to inference. Unfortunately, finding the optimal ordering of inputs entails search through the permutation space of variables. Furthermore, in real-world applications of structure learning, the gold standard network is typically unknown. In this paper, we first present a genetic algorithm (GA) that uses a well-known greedy algorithm for structure learning (*K2*) and approximate inference by importance sampling as primitives in searching this permutation space. We then develop a flexible fitness measure based upon inferential loss given a specification of evidence. Finally, we evaluate this GA wrapper using the well-known networks *Asia* and *ALARM* and show that it is competitive with exhaustive enumeration in finding good orderings for *K2*, resulting in structures with low inferential loss under importance sampling.

Keywords: Bayesian networks, genetic algorithms, permutation problems, probabilistic reasoning, machine learning, wrappers

1 INTRODUCTION

Learning the structure, or causal dependencies, of a graphical model of probability such as a Bayesian network (BN) is often a first step in reasoning under uncertainty. In many machine learning applications, it is therefore referred to as a method of *causal discovery* [PV91]. Finding the optimal structure of a BN from data has been shown to be *NP*-hard [HGC95], even without considering latent (unobserved) or irrelevant (extraneous) variables. Therefore, greedy *score-based* algorithms [FG98] have been developed to provide more efficient structure learning at an accuracy tradeoff. In this paper we examine a general shortcoming of greedy structure learning – sensitivity to variable ordering – and develop a genetic algorithm to mitigate this problem by searching the permutation space of variables using a probabilistic inference criterion as the fitness function.

We make the case in this paper that the probabilistic inference performance element, **in the absence of a known gold standard network** or any explicit constraints, can provide the feedback needed to search for a good ordering. We then derive a heuristic based on validation by inference (exact inference [LS88, Ne90] for small networks, approximate inference by stochastic sampling [CD00] for larger ones). Our primary objective is inferential accuracy *using* the learned structure.

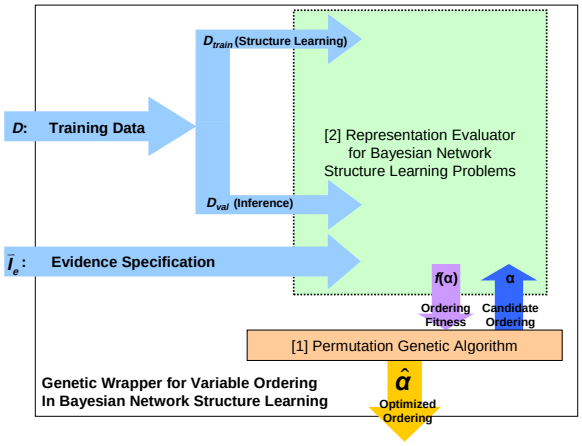


Figure 1. System Design Overview.

Toward this end, we adapt a flexible, composite fitness measure used in other machine learning systems called *wrappers* [KJ97], which automatically tune hyperparameters of the learning system such as the ordering of input variables. We present the system shown in Figure 1, a genetic algorithm-based wrapper [CS96, RPG+98, HWRC01], and show how it provides a parallel stochastic search mechanism for inferential loss-minimizing variable orderings. We demonstrate that, used in tandem with *K2*, it produces structures whose loss under importance sampling is nearly as low as any found by exhaustive enumeration of orderings. Finally, we discuss how this wrapper provides a flexible method for tuning *representation biases* [Mi97] in Bayesian network structure learning using different fitness criteria.

2 VALIDATION OF STRUCTURES

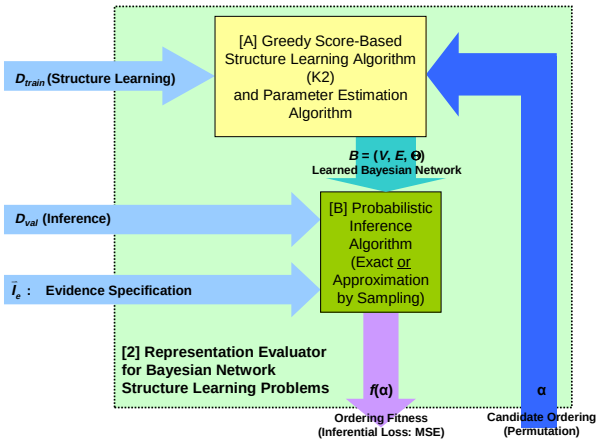


Figure 2. Probabilistic reasoning environment, Module [2] from Figure 1.

Consider a typical probabilistic reasoning environment, as shown in Figure 2, where structure learning [A] is a first step. The input to this system includes a set D of training data vectors $\mathbf{x} = (x_1, \ldots, x_n)$ each containing n variables. If the structure learning algorithm is greedy, an ordering $\boldsymbol{\alpha}$ on the variables may also be given as input. The structure learning component of this system produces a graphical model $B = (V, E, \Theta)$ that describes the dependencies among X_i, including the conditional probability functions. The inferential performance element [B] of this system takes B and a new data set D_{test} of vectors drawn from the desired inference space, where only a subvector $\mathbf{E}$ of $\mathbf{X} = (X_1, \ldots, X_n)$ is observable, and infers the remaining unobserved values $\mathbf{X} \setminus \mathbf{E}$. We denote the indicator bit vector for membership in $\mathbf{E}$ by $\mathbf{I_e}$. The performance criterion f is the additive inverse of the (inferential or utility) loss of [B].

This section specifies the functionality of [A] and [B] and explains the derivation of f as a function of the ordering $\boldsymbol{\alpha}$. In the next section, we show how the environment depicted in Figure 2 is used as the fitness evaluation module [2] of the overall GA-based system (Figure 1).

2.1 Learning Bayesian Network Structure

Consider a finite set $\chi = \{X_1, \ldots, X_n\}$ of discrete random variables. A *Bayesian network* is an annotated directed acyclic graph $G = (V, E)$ that encodes a joint probability distribution over χ. The nodes of the graph correspond to the random variables $X_1, \ldots, X_n$. Each node is annotated with the conditional probability distribution (CPD) that represents $P(X_i \mid Pa_{xi})$, where Pa_{xi} denotes the parents of X_i in G. A Bayesian network B specifies the unique joint probability distribution over χ given by:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P\left(X_i \mid Pa_{x_i}\right) \quad (1)$$

The graph G represents conditional independence properties of the distribution. These are the *Markov independencies*: each variable X_i is independent of its non-descendants, given its parents, in G. [EF01] We denote the annotating CPD parameters of B by Θ; thus, $B = (V, E, \Theta)$.

We are interested in learning B from training data D consisting of examples $\mathbf{x}$. For simplicity, we assume that there are no variables that are latent or completely irrelevant (not weakly relevant [KJ97]). The objective of structure learning is then to find the arcs E for $V = \chi$. Some structure learning algorithms, such as *K2* [CH92], are greedy in that they add arcs based upon the incremental gain that each single arc induces in a global score, such as the Bayesian (Dirichlet) score. [CH92, FG98]. We use *K2* for structure learning – module [A] of Figure 2 – because it finds structures quickly *if* given a reasonable ordering $\boldsymbol{\alpha}$. Variables must occur "upstream" from one another (or "downstream" in $\boldsymbol{\alpha}$, i.e., have a higher index) to be considered as candidate parents. If the number of parents per variable is constrained to a constant upper bound, *K2* has worst-case polynomial running time in the number n of variables.

Two clear limitations of greediness are inability to backtrack (i.e., undo the addition of an arc) or consider the joint effects of adding multiple arcs (parents). This is why greedy structure learning algorithms are sensitive to the presence of irrelevant variables in the training data, a pervasive problem in machine learning [KJ97]. Additionally, *K2* is particularly sensitive to the variable ordering because arcs fail to be added, resulting in unexplained correlations, whenever candidate parents are evaluated in any order that precludes a causal dependency. Were a gold standard structure $G^* = (V, E^*)$ available, this would be seen as an inversion in the partial ordering induced by E^*. Preventing missing arcs – i.e., "false negatives for causality" – is a challenge in structure learning as applied to causal discovery [PV91, FG98].

Unfortunately, just as finding the optimal structure is itself intractable [HGC95], so is finding the optimal ordering of inputs for a given structure learning algorithm. Searching the space of permutations of variables is prohibitive, and defeats the purpose of using a greedy algorithm. In this paper, we focus on *K2* and the problem of optimizing the variables to be given as its input. To specify the optimization of variable order as a search problem, we must define the states (permutations), operators (re-ordering), initial candidates, and evaluation criterion.

2.2 Validation by Inference

A desired joint probability distribution function $\mathbf{P}(\mathbf{X})$ can be computed using the chain rule for Bayesian networks, given above in Equation (1). The *most probable*

explanation (*MPE*) is a truth assignment, or more generally, value assignment, to a *query* **Q** = **X** \ **E** with maximal posterior probability given evidence **e**. Finding the MPE directly using Equation (1), requires enumeration of exponentially many explanations. Instead, a family of exact inference algorithms known as *clique-tree propagation* (also called *join tree* or *junction tree* propagation) is typically used in probabilistic reasoning applications. The first of these algorithms was developed by Lauritzen and Spiegelhalter [LS88, Ne90]. Although exact inference is important in that it provides the only completely accurate baseline for the fitness function *f*, the problem for general BNs is *#P*-complete (thus, deciding whether a particular truth instantiation is the MPE is *NP*-complete) [Co90, Wi02].

Approximate inference refers to approximation of the posterior probabilities given evidence. One stochastic approximation method called *importance sampling* [CD00] estimates the evidence marginal by sampling query node instantiations:

$$\mathbf{P(E = e)} = \sum_{\mathbf{X \setminus E}} \mathbf{P}\left(\mathbf{X \setminus E} \mid \mathbf{E} = \mathbf{e}\right) \tag{2}$$

[CD00] discusses basic variants of importance sampling. These include *probabilistic logic sampling* [He86], whose importance function is the joint distribution function **P**(**X**). By sampling from the network as if no evidence were given, the priors on source or root nodes are emphasized, resulting in a possibly suboptimal importance function as the authors point out. The source priors are similarly emphasized in *forward simulation* by likelihood weighting [SP89, CD00], which samples using the joint probability of query nodes as the importance function:

$$\mathbf{P(X \setminus E)} = \sum_{x \notin e} P\left(x_i \mid Pa_{x_i}\right) \tag{3}$$

Welch demonstrates [We96] that even a moderately complex binary network with deterministic nodes, approximately the size of *ALARM*, can be difficult to sample from by pure forward sampling if there are enough query nodes (evidence) – the author instantiates 4 of 32 binary nodes with a moderately unlikely evidence vector, **P(e)** = $6.5 * 10^{-4}$.

One way of scaling up to large networks in a realistic probabilistic reasoning application is to dynamically adapt the importance function. [CD00] presents a solution of this type called *adaptive importance sampling* (*AIS*), where a dynamic importance function is first initialized using structural heuristics, then empirically trained in each of several training steps. This is similar to the hyperparameter sampling stages in Markov chain Monte Carlo (**MCMC**) methods [Ne93]. The key issue is whether we have any prior knowledge regarding the estimators (e.g., heuristic importance functions).

We have implemented five variants of importance sampling: forward simulation, logic (*aka* rejection) sampling, backward sampling, self and heuristic importance sampling, and adaptive importance sampling. Because adaptive importance sampling has been empirically shown [CD00] to be more robust in the presence of unlikely evidence **e**, and because we have found it to converge quickly in independent experiments, we use it in our evaluation component, module [B] in Figure 2 above.

2.3 Deriving Fitness

To optimize the ordering, we considered fitness functions with three objective criteria. In this paper, however, we focus *solely* on the first:

1. **Inferential loss**: Quality of the network produced by *K2* as detected through inferential loss evaluated over a holdout validation data set $D_{val} \equiv D \setminus D_{train}$ (see Figure 1) – requires modules [A] and [B] in Figure 2
2. **Model loss**: "Size" of the network under a specified representation – requires module [A] only and is independent of [B]
3. **Ordering loss**: Inference and model-independent measure of data quality given only *D* and **α** – independent of both modules [A] and [B]

$$f\left(\boldsymbol{\alpha}, D, \vec{\mathbf{I}}_{\mathbf{e}}\right) = a \cdot f_a\left(\boldsymbol{\alpha}, D, \vec{\mathbf{I}}_{\mathbf{e}}\right) + b \cdot f_b\left(\boldsymbol{\alpha}, D\right) + c \cdot f_c\left(\boldsymbol{\alpha}, D\right) \tag{4}$$

$$f_a\left(\boldsymbol{\alpha}, D, \vec{\mathbf{I}}_{\mathbf{e}}\right) = 1 - \sqrt{\frac{1}{\sum_{X_i \in \mathbf{X} \setminus \mathbf{E}} a_i} \sum_{X_i \in \mathbf{X} \setminus \mathbf{E}} \sum_{j=1}^{a_i} \left(P'\left(x_{ij}\right) - P\left(x_{ij}\right)\right)^2} \tag{5}$$

$$f_b\left(\boldsymbol{\alpha}, D\right) = 1 - \frac{\sum_{i=1}^{n} \left(a_i \cdot max\left(\prod_{\left|X_j \in \mathbf{Pa}_{x_i}\right|} a_j, 1 \right) \right)}{\prod_{i=1}^{n} a_i} \tag{6}$$

$$\textbf{\textit{where}} \quad a_i \equiv arity\left(X_i, B = \left(\chi, E, \Theta\right)\right)$$
$$\left(E, \Theta\right) = K2\left(\boldsymbol{\alpha}, D_{train}\right)$$
$$a + b + c = 1 \tag{7}$$

In related work on genetic wrappers for variable selection in supervised inductive learning, Hsu *et al* adapted Equation (4) [HWRC00, HWRC01] from similar fitness functions developed by Cherkauer and Shavlik for decision tree pre-pruning [CS96], Raymer *et al* for similarity-based learning (*k*-nearest neighbor regression) [RPG+97], and Whitley and Guerra-Salcedo for connectionist learning [GW99]. This breadth of applicability demonstrates the generality of simple genetic algorithms as wrappers for performance tuning in supervised inductive learning.

Recently, Hsu *et al* automatically validated the coefficients *a*, *b*, and *c* for several individual data sets on a supervised learning task. [HWRC02] Results were positive in that this approach found application-specific values for these *hyperparameters*, and the GA achieved better generalization accuracy than search-based feature selection wrappers [KJ97] for a real-world test bed (prediction of loss ratio in automobile insurance risk

analysis). Controlling the values of *a*, *b*, and *c* simultaneously proved to be difficult in that large amounts of validation data were required, and the authors report that experiments did not indicate conclusively whether the GA performed better with this single composite-objective fitness function or a multi-objective one (i.e., Pareto optimization). Therefore, for clarity, **we set b and c to 0 to ignore f_b and f_c** in the experiments reported in this paper. In the last section, we discuss the ramifications of this design choice and possible future work using the full *f*.

We now focus on the first term, f_a. This fitness function computes inferential loss by measuring the predictive power of the Bayesian network on the data set given a specification of evidence, $\mathbf{I_e}$. The specific f_a we use is the normalized additive inverse of the root mean squared error (RMSE), which is the square root of the sum of squared differences betweeen the sampled, approximate probabilities $P'(x_{ij})$ and exact probabilities $P(x_{ij})$, over states x_{ij} of variables X_i. [CD00] Note that f_a is the only term that depends on which variables are *observable***,** i.e., members of **E**. We consider this the most important term just as validation set classification error is considered a typical estimator of generalization error in supervised classification learning [Mi97]. Ultimately, a BN *B* is only as good as the inferences it can produce on real-world data given realistic evidence **e**, and an ordering $\boldsymbol{\alpha}$ is only as good as the BN that it can induce given a specific structure learning algorithm. In the next section, we explain why this is a motivation for GA wrappers in general.

3 SEARCH-BASED ENHANCEMENT OF LEARNING

Figure 1 indicates the role of a combinatorial optimization system for controlling $\boldsymbol{\alpha}$, in context: a probabilistic reasoning system based on greedy structure learning can use an optimized ordering $\hat{\boldsymbol{\alpha}}$ to enhance structure quality. This is done by searching for a good $\boldsymbol{\alpha}$ using a "realistic" inferential criterion and a fixed, greedy structure learning algorithm such as *K2*. We now explore this combinatorial optimization problem and the design of our specific GA.

3.1 Wrapper Approaches to Optimizing Input

Tuning machine learning algorithms for large, complex data sets is an expensive and difficult task. In addition to identifying the appropriate inputs for a particular classification or inference performance element, the system designer must find a representation for hypotheses, i.e. the language for expressing the target concept, and a suitable performance measure by which to evaluate hypotheses. Making appropriate decisions regarding the input specification is crucial for tractable learning, because these determine part of the *inductive bias* [Be90, Mi97] of the learning system. *Bias*, the preferences of a learning system for one hypothesis over another other than those dictated by consistency with the training data, determines how the space of hypotheses (in our application, BN structures) is to be searched and can radically affect the tractability of this search. Unfortunately, effective decisions often depend in subtle ways upon the learning algorithm, training data, and their interaction. A mechanism for systematically identifying good inputs should take the performance element of the system input into account.[1] It must have the ability to tune the learning system by automatically adjusting the some aspect of the input specification (e.g., selected variables, *aka feature subsets*, or variable orderings $\boldsymbol{\alpha}$) and coefficients for quantitative inductive bias such as those discussed previously. Controlling all of these parameters, while keeping the machine learning system efficient and manageable, is not easy.

We approach this problem in BN structure learning by applying search-based combinatorial optimization and use *validation by inference* (presented in the previous section) as a search heuristic. The high-level mechanisms that determine a learning system's representation and preference biases can be expressed using learning *hyperparameters* [Ne93], such as $\boldsymbol{\alpha}$. Just as a learning parameter denotes a trainable component of a pattern detector or classification function, a learning hyperparameter denotes a controllable component of the organization, representation, or search algorithm for a learning problem. Inductive learning systems, or *inducers*, are built with such hyperparameters and the ability to tune them using combinatorial search, based upon evaluation metrics over validation data. The benefits to probabilistic learning and reasoning are the potential for greater flexibility in learning processes, an increase in generalization quality, and the ability to make the learning component more automatic and transparent.

3.2 GA-Based Wrappers

A GA is ideal for implementing wrappers where hyperparameters are naturally encoded as chromosomes such as bit strings or permutations. This is precisely the case with variable (feature subset) selection, where a bit string can denote membership in the subset, and with variable ordering, where a permutation denotes $\boldsymbol{\alpha}$, the order in which nodes are added to the BN. Both of these are forms of *constructive induction* where the input representation is changed from the default [Be90] – here, the full subset χ or an arbitrary ordering $\boldsymbol{\alpha_0}$.

With a GA-based wrapper, we seek to evolve hyperparameter values using the performance criterion of

[1] The term *wrapper* as used in machine learning [Ko95, KJ97] simply refers to this property, wherein the combinatorial optimization system "wraps around" a specific inductive learning and classification or inference ensemble such as the one shown in Figure 2. In the genetic and evolutionary computation literature, as we note below, wrappers for tuning GA hyperparameters have been in use for quite some time. [BGH89, DSG93, HL99]

the overall learning system as fitness. In learning to classify, this may simply mean validation set accuracy. However, as we have noted, many authors of GA-based wrappers have independently derived criteria that resemble *minimum description length* (*MDL*) estimators – that is, they seek to minimize model size and the sample complexity of input as well as maximize generalization accuracy. [CS96, RPG+97, GW99, HWRC00]

An additional benefit of GA-based wrappers is that it can automatically calibrate "empirically determined" constants such as the coefficients *a*, *b*, and *c* introduced in the previous section. As we noted, this can be done using individual training data sets rather than assuming that a single optimum exists for a large set of machine learning problems. This is preferable to empirically calibrating hyperparameters as if a single "best mixture" existed. Even if a very large and representative corpus of data sets were used for this purpose, there is no reason to believe that there is a single *a posteriori* optimum for hyperparameters such as weight allocation to inferential loss, model complexity, and sample complexity of data in the constructive induction wrapper.

Finally, GA wrappers can "tune themselves" – for example, the *GA-Based Inductive Learning* (*GABIL*) system of Dejong *et al* [DSG93] learns propositional rules from data and adjusts constraint hyperparameters that control how these rules can be generalized. Mitchell notes that this is a method for evolving the learning strategy itself. [Mi97] Many classifier systems also implement performance-tuning wrappers in this way. [BGH89] Finally, population size and other constants for controlling elitism, niching, sharing, and scaling can be controlled using *parameterless GAs*. [HL99]

We adapted *GAJIT* [Fa00], a Java shell for developing genetic algorithms, to implement a GA for the permutation problem of ordering variables for Bayesian network structure learning (using *K2*) and inference (using the Lauritzen-Spiegelhalter algorithm [LS88, Ne90] and *forward simulation* [SP89, CD00]). We now specify the ordering problem and, in the next section, present the permutation GA design.

3.3 Ordering and Structure Learning Problems

The ordering problem itself is a straightforward search in permutation space $\mathbf{A}$ for a value of $\boldsymbol{\alpha}$ that minimizes the inferential loss or maximizes its normalized, additive inverse, f_a. Some simple combinatorial analysis illustrates the relative complexity of the ordering and structure learning problems.

Clearly $|\mathbf{A}| = n!$ if we suppose that there are no latent or irrelevant variables. From Stirling's approximation, we can estimate that $|\mathbf{A}| \approx 2^{n \lg n}$. Meanwhile, we know that all elements of structure space are directed acyclic graphs, containing some subset of the n^2 possible directed edges. The size of structure space is thus in $O(2^{n^2})$. Note that this includes all directed graphs and is therefore an overestimate. Taking the asymptotic ratio of these two counting functions, however, we see that in the limit, there are infinitely many possible structures *for each* ordering. *K2*, which is deterministic, finds just one such structure, so it is not guaranteed that finding a loss-minimal ordering $\boldsymbol{\alpha}$ will cause it to produce a loss-optimal network *B*, particularly for very large *n*. However, Friedman conjectures [FLNP00] that searching ordering space provides a useful change of representation [Be90] that tends to admit smoother interpolation than in structure space. In evolutionary computation terms, this would mean that ordering space is less *deceptive* [Go89] than structure space.

4 GA FOR VARIABLE ORDERING

4.1 Searching Ordering Space

The criterion f_a is computed by actually learning a BN $B = K2(\boldsymbol{\alpha}, D_{train})$ – more precisely $(E,\Theta) = K2(\boldsymbol{\alpha}, D_{train})$.

E is computed by *K2*, which makes a single pass through $\boldsymbol{\alpha}$ (a permutation of $\chi = \{X_1, \ldots, X_n\}$) and, for each X_i, considering only X_j where $\boldsymbol{\alpha}(j) > \boldsymbol{\alpha}(i)$ as a potential parent of X_i in *E*. It then adds X_j to Pa_{xi} by adding (X_j, X_i) to *E* if and only if this increases the Dirichlet score of Pa_{xi}, evaluated over D_{train}. This continues until: the set of X_j is exhausted, no single parent can be added to incrementally increase the score, or a preset (or automatically calibrated) limit on the size of Pa_{xi} in *E* is reached. For discrete BNs, Θ is computed simply by populating the specified conditional probability tables (CPTs) with frequencies computed using D_{train}.

Once *B* is fully learned, each example in $D_{val} \equiv D \setminus D_{train}$ is masked with $\mathbf{I}_e$ and its complement to obtain separate evidence and query data. The inferential loss f_a is computed as specified in the previous section. The ordering problem is a combinatorial search in $\mathbf{A}$ using f_a as a heuristic.

4.2 Permutation Genetic Algorithm Design

Application of genetic algorithms to permutation problems is discussed in [Go89] and [HH99]. The design of the *GAJIT* wrapper illustrated in Figure 1 is as follows.

We implemented an elitist permutation GA purely by extending the *GAJIT* classes using order crossover (OX) [HH99]. OX exchanges subsequences of two permutations, displacing duplicate indices with holes. It then shifts the holes to one side, possibly displacing some indices, and replaces the original subsequence in these holes. If two parents p_1 = [3 4 6 2 1 5] and p_2 = [4 1 5 3 2 6] are recombined using OX, with the crossover mask underlined, the resulting intermediate representation is i_1 = [- - 5 3 1 4] and i_2 = [- - 6 2 4 1], and the offspring are o_1 = [6 2 5 3 1 4] and o_2 = [5 3 6 2 4 1]. Mutation is implemented by swapping uniformly selected indices. *Cataclysmic mutation* [GW99] can easily be implemented using a *shuffle* operator, but we did not find this necessary.

The *master controller* for our GA runs in a Java virtual machine. It manages slaves that concurrently evaluate members of its population **α**. Each individual is encoded as a permutation of the indices {1, …, *n*}. Slave processes distributed across (4-48 processors) of a distributed-shared memory (DSM) compute cluster run identical copies of the *K2* and inference-based application depicted in Figure 2. Each evaluates the ordering it is given by learning *B* from D_{train}, a holdout segment of *D* (2/3 by default) and returns f_a for the validation set $D_{val} \equiv D \setminus D_{train}$. The master GA collects the fitness components for all members of its population and then computes *f* (here, $f = f_a$).

5 EXPERIMENTAL RESULTS AND EVALUATION

We experimented using the GA on data simulated from the well-known toy BN *Asia* [Ne90], which has 8 nodes. This is a very simple network to perform inference on when the structure is known *a priori*, but the permutation space – which we are searching using only *f* and the synthetic data – has 8! = 40320 orderings. We also performed exploratory experiments using two versions of the *ALARM* network: a subgraph of 13 nodes and the full 37-node network.

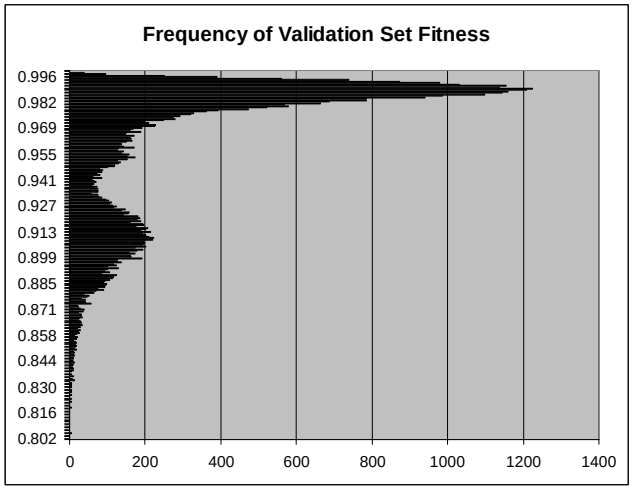


Figure 3. Histogram of estimated fitness for all 8! = 40320 permutations of *Asia* variables.

Figure 3 depicts the histogram of validation set fitness as measured **exhaustively** using Equation 5 and forward simulation [SP89, CD00]. Each of the 8! = 40320 fitness evaluations was made by running *K2* on D_{train} (as shown in Figure 2), consisting of 20000 stochastically-generated samples, and then evaluating the resulting BN using forward simulation on D_{test} (a *holdout test set* **not** used by the GA as D_{val} in Figure 2) and an evidence bit vector $\mathbf{I_e}$= (1 0 0 0 0 0 0 1). The histogram shown corresponds to data generated from the evidence instantiation *Visit-to-Asia = true* ∧ *Dyspnoea = false*. We note that this is just one evidence specification among many plausible ones that might occur in "real" applications of this consultative BN. The mean fitness is 0.958, the range is [0.0802, 0.999], and the standard deviation is 0.039.

Table 1 summarizes experimental specifications using the experimental platform described in the previous section. Figure 4 shows the average-fitness curve for *Asia* using the *GAJIT* wrapper. Using forward simulation [SP89, CD00], we generated 20000 samples for D_{train}, 5000 for D_{val} (used to evaluate fitness in the GA), and 5000 for D_{val} (used to evaluate "generalization fitness" on the ordering returned by the GA). The number of stochastic samples used to perform inference on D_{val} is given in Table 1; for all runs, 15000 samples were used to perform inference on D_{train}. The GA uses OX (order crossover), swap-mutation, and a population of size 10, and was run for 100 generations.

K2	FS Samples	Best *f* of final gen
5000	1500	0.944
10000	1500	0.960
20000	*150*	*0.935*
20000	*450*	*0.977*
20000	***1500***	***0.978***

Table 1. Results for *Asia* (5000 samples per fitness evaluation in D_{val} and D_{test})

The results for the last line were averaged over 3 trials but Figure 4 depicts the median result. Starting from a test fitness of 0.4 (inferential loss of 0.6), it improves the test fitness to 0.98. This is only about slightly above the mean fitness but it is noteworthy that the gold standard network achieves fitness of only 0.98 as well. We validated this using exact inference (the Lauritzen-Spiegelhalter algorithm [LS88, Ne90]) to compute the marginals on the data and our forward simulation function itself converges to negligibly low relative loss.

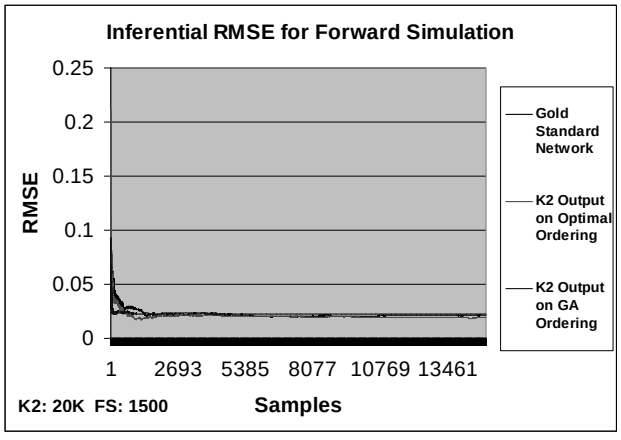


Figure 4. Fitness curve for last run in Table 1

As the fitness curve shows, the *GAJIT* wrapper reaches 0.98 rather quickly. The highest fitness achieved by the wrapper on any run is 0.99, and inspection shows that the corresponding ordering has only one inversion from the canonical one given by Neapolitan [Ne90]. This inversion is consistent with the partial ordering of the

canonical *B*, which means that *K2* can still produce the best possible structure from it.

Experiments using *ALARM-13* and *ALARM-37* indicated that although *K2* is capable of recovering a graph (*V*, *E*) close to the gold standard network (Cooper and Herskovits report only two graph errors using only 20000 training examples [CH92], as we used), its algorithm for estimating conditional probability tables results in high relative inferential loss. We hypothesize that this is due to the skewness of some conditional probability tables (CPTs) in both versions of *ALARM*. The fitness evaluation procedure depicted in Figure 2 is therefore is less effective than on *Asia*. In continuing work, we are hybridizing *K2* with other CPT learning algorithms.

6 DISCUSSION AND FUTURE WORK

We have considered several continuations of this research, grouped into four categories: validation, scalability, comparison to other structure learning methods, and improvements to the ordering GA.

First, validation is currently performed by running importance sampling for precisely 15000 samples (with an importance function update every 100 samples for AIS), and this is repeated to find the fitness of the best ordering $\hat{\boldsymbol{\alpha}}$ found by the generational GA. Experiments currently in development run *K2* with a range of D_{train} sizes to generate a learning curve, and run AIS longer with $\hat{\boldsymbol{\alpha}}$ to get a more accurate evaluation. Automated convergence analysis can be used to adapt the number of samples and the AIS update rate. Fast exact inference to find the true inferential loss baseline, a topic of a concurrent research project, can test the efficacy of AIS itself. We have focused in this paper on the general case, where the gold standard network may not be known, but when it is, one can use graph edit distance between the BN induced by $\hat{\boldsymbol{\alpha}}$ and the gold standard as a validation measure [CH92].

Second, we plan to explore the scalability of the GA wrapper by experimenting with larger networks (such as *ALARM* and *Pathfinder*) with which we have already tested AIS and *K2* as individual components. When used in a GA, which may evaluate fitness thousands to millions of times for this problem, these primitives to become bottlenecks. To make the wrapper feasible, it will be necessary to parallelize *K2* and AIS.

Third, there are several algorithms besides greedy search for structure learning, such as deterministic score-based (sparse candidate, Tabu search) methods, constraint-based methods, stochastic sampling in structure space by direct (non-greedy) global optimization and stochastic sampling in ordering space (to determine structure, without using a greedy algorithm such as K2 as an intermediary). These are often less sensitive to variable ordering but may still be affected by it. In continuing work, we plan to compare our GA wrapper to these techniques.

Fourth, the following are promising variants of the GA that are high experimental priorities: Pareto optimization of (f_a, f_b, f_c) and experimentation with other permutation mutation and crossover operators (partially matched and cycle crossover).

Acknowledgements

Support for this research was provided in part by the Army Research Lab under grant ARL-PET-IMT-KSU-07 and by the Office of Naval Research under grants N00014-00-1-0769 and N00014-01-1-0917.

References

[Be90] D. P. Benjamin, editor. *Change of Representation and Inductive Bias*. Kluwer Academic Publishers, Boston, 1990.

[BGH89] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40:235-282, 1989.

[CD00] J. Cheng and M. J. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. Journal of Artificial Intelligence Research (JAIR), 13:155-188, 2000.

[CH92] G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. Machine Learning, **9**(4):309-347, 1992.

[Co90] G. F. Cooper. The computational complexity of probabilistic infernece using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393-405. Elsevier, 1990.

[CS96] K. J. Cherkauer and J. W. Shavlik. Growing Simpler Decision Trees to Facilitiate Knowledge Discovery. In *Proceedings of the Second International Conference of Knowledge Discovery and Data Mining (KDD-96)*, Portland, OR, August, 1996.

[DSG93] K. A. DeJong, W. M. Spears, and D. F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161-188, Kluwer Academic Publishers, 1993.

[EF01] G. Elidan and N. Friedman. Learning the Dimensionality of Hidden Variables. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, Morgan-Kaufmann, 2001.

[FG98] N. Friedman and M. Goldszmidt. *Learning Bayesian Networks From Data*. Tutorial, American National Conference on Artificial Intelligence (AAAI-98), Madison, WI. AAAI Press, San Mateo, CA, 1998.

[FLNP00] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, Using Bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology (RECOMB 2000)*, ACM-SIGACT, April 2000.

[Go89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[GW99] C. Guerra-Salcedo and D. Whitley. Genetic Approach to Feature Selection for Ensemble Creation. In *Proceedings of the 1999 International Conference on*

Genetic and Evolutionary Computation (GECCO-99). Morgan-Kaufmann, San Mateo, CA, 1999.

[HGC95] D. Heckerman, D. Geiger, and D. Chickering, Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197-243, Kluwer, 1995.

[HH98] R. L. Haupt and S. E. Haupt. *Practical Genetic Algorithms*. Wiley-Interscience, New York, NY, 1998.

[HL99] G. Harik and F. Lobo. *A parameter-less genetic algorithm.* Illinois Genetic Algorithms Laboratory technical report 99009, 1999.

[HWRC00] W. H. Hsu, M. Welge, T. Redman, and D. Clutter. Genetic Wrappers for Constructive Induction in High-Performance Data Mining. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Las Vegas, NV. Morgan-Kaufmann, San Mateo, CA, 2000.

[HWRC02] W. H. Hsu, M. Welge, T. Redman, and D. Clutter. Constructive Induction Wrappers in High-Performance Commercial Data Mining and Decision Support Systems. *Knowledge Discovery and Data Mining*, Kluwer, 2002.

[KJ97] R. Kohavi and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence, Special Issue on Relevance, 97*(1-2):273-324, 1997.

[Fa00] M. Faupel. GAJIT genetic algorithm package. URL: http://www.angelfire.com/ca/Amnesiac/gajit.html, 2000.

[Mi97] T. M. Mitchell. *Machine Learning.* McGraw-Hill, New York, NY, 1997.

[LS88] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B* 50, 1988.

[Ne90] R. E. Neapolitan. Probabilistic Reasoning in Expert Systems: Theory and Applications. Wiley-Interscience, New York, NY, 1990.

[Ne93] R. M. Neal. Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

[PV91] J. Pearl and T. S. Verma, A theory of inferred causation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*. Morgan Kaufmann, San Mateo, CA, 1991.

[RPG+97] M. Raymer, W. Punch, E. Goodman, P. Sanschagrin, and L. Kuhn, Simultaneous Feature Extraction and Selection using a Masking Genetic Algorithm, In *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 561-567, San Francisco, CA, July, 1997.

[SP89] R. D. Schacter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence 5*, p. 221-231, Elsevier Science Publishing Company, New York, NY, 1989.

[We96] R. L. Welch. Real-Time Estimation of Bayesian Networks. In *Proceedings of UAI-96*, Morgan-Kaufmann, 1996.

[Wi02] Wikipedia Online Encyclopedia, *Sharp-P*. URL: http://www.wikipedia.com/wiki/Sharp-P, 2002.

Balancing Learning and Evolution

Michael Hüsken **Christian Igel**
Institut für Neuroinformatik
Ruhr-Universität Bochum
44780 Bochum, Germany
{michael.huesken,christian.igel}@neuroinformatik.ruhr-uni-bochum.de

Abstract

Finding the right coupling of learning and evolution in a hybrid algorithm is an open problem. In this article, we present a strategy to adjust the time spent on learning during evolutionary optimization of neural networks. The proposed adaptation scheme leads to a significant improvement in performance. It is empirically shown that suitable learning strategies strongly depend on the problem and that it is advantageous to adapt the time spent on learning during evolution.

1 INTRODUCTION

Evolution and learning are biologically-inspired optimization paradigms that have proven to be efficient in a wide range of applications, particularly when—as in nature—combined in hybrid strategies. Both techniques have different characteristics and their coupling aims at getting the best of both worlds. Evolution and learning interact in complex ways (e.g., see Mayley, 1996; Nolfi and Parisi, 1999). Further, it appears to be obvious that the optimal way of combining them depends on the target problem and may change during optimization. However, a systematic way to balance the two strategies is still missing. In this article, we present an adaptive method for controlling the time spent on learning in the framework of evolutionary optimization of neural networks.

Structure optimization of artificial neural networks is the most prominent technical example of a successful combination of evolution and learning, where evolution is employed to optimize mainly the network's architecture and learning (or training) is usually identified with gradient-based adaptation of the weights (Yao, 1999). In this paper, we assume that learning and evolution interact in a typical manner: Each offspring individual, representing one network architecture with the corresponding initial weights, is trained once immediately after its creation. The fitness of the individual is determined after training.

Most learning algorithms are iterative methods and their computational complexity usually scales linearly with the number of iterations. When searching for a neural network for a given task, several questions arise: Is learning necessary or is evolution sufficient? If learning is employed, how should evolution and learning be coupled? How long should each system learn? We address the last question, which might lead to an answer to the first one. For several reasons the learning time is a crucial parameter, e.g.:

- In most cases, the computational costs for generating and evaluating an offspring can be neglected compared to the costs for training. For standard neural network architectures, even a single learning iteration typically takes about twice the time of one fitness evaluation.
- If the learning time is too long, the neural network may overfit and lose its ability to generalize, i.e., to react in a desired way to data not used for learning. Even if no overfitting occurs, long learning might be a waste of resources in cases of bad local minima or insufficient architectures.
- If the maximum number of iterations is too small, the weights may not have enough time to adapt to an altered architecture. Hence, better architectures may be discarded, because the learning time is too short to uncover their benefits.

It is intuitive—and will become obvious in this study—that the right learning time depends on the problem at hand and on the course of evolution. Nevertheless,

no systematic way exists either to choose the learning time for a given task or to adjust it during evolution.

In this article, we present a strategy to adapt the time spent on learning. The basic idea is inspired by strategy adaptation methods in evolutionary computation (Smith and Fogarty, 1997; Eiben et al., 1999). The learning time is viewed as a strategy parameter that is adjusted on population level. As a basis for adaptation, different numbers of iterations have to be explored. Therefore, the learning time is described by a random variable. The maximum number of iterations an offspring is allowed to learn is a realization of this random variable, whose expectation is subject to adaptation: The expected learning time in the upcoming generations is shifted towards the learning time that gave good results recently.

We proceed with the presentation of a straightforward algorithm for evolutionary structure optimization of neural networks and with the detailed explanation of the adaptation scheme for the learning time. In section 3, we present experimental results showing the dynamics of the learning times induced by our adaptation scheme as well as the improved optimization performance. The paper ends with concluding remarks.

2 LEARNING TIME ADAPTATION IN STRUCTURE OPTIMIZATION

In the main part of this section, we introduce the adaptation scheme for the learning time. Beforehand, we describe a simple algorithm for structure optimization of neural networks, which serves as a testbed for our method. However, the proposed adaptation scheme is designed to work well with almost any of such algorithms.

2.1 EVOLUTIONARY FRAMEWORK

We use an evolutionary algorithm for structure optimization of neural networks inspired by Angeline et al. (1994). The search space contains all arbitrarily connected feed-forward neural networks. The validity of the architectures is the only constraint, i.e., each hidden neuron has to lie on a path from at least one input to at least one output neuron. Each of the μ individuals in the population encodes the architecture and the weights of one neural network by means of a direct encoding scheme. Prior to the first generation, the individuals are randomly initialized and assigned a fitness value, depending on the target problem. In every generation, each parent produces one offspring by reproduction and mutation. One out of five problem specific mutation operators is randomly chosen and applied: adding and deleting single neurons and single connections as well as disturbing each weight normally distributed with zero mean and standard deviation σ; connections with a lower weight are removed with an increased probability (Braun, 1997). Thereafter, each individual is trained using the iRprop$^+$ learning method (Igel and Hüsken, 2002), an improved version of the well known Rprop algorithm (Riedmiller and Braun, 1993). Learning starts with the genetically encoded weight configuration. After learning the modified weights are inherited, i.e., coded back onto the individual's genome, following the Lamarckian paradigm, which is very efficient for technical purposes. Finally, the individual's fitness is evaluated and the parent population of the next generation is determined by means of EP-tournament selection (Fogel, 1995).

2.2 ADAPTATION OF THE LEARNING TIME

The aim of adjusting the learning time is to choose its value such that in the next generations a maximum fitness improvement relative to the costs can be expected. This task is similar to the adaptation of the strategy parameters of an evolutionary algorithm (e.g., population size, mutation rates, ...), which is a key concept in evolutionary computation (Smith and Fogarty, 1997; Eiben et al., 1999). Our scheme is inspired by these methods, in particular by the covariance matrix adaptation (CMA) evolution strategy (Hansen and Ostermeier, 2001). The adjustment of the learning time is based on the heuristic that learning times that have given good results recently will also perform well in future generations.

The learning strategy is altered every G generations; this interval is called *adaptation cycle*. For the adaptation of the strategy (i.e., the average learning time in the g^{th} adaptation cycle), it is necessary to explore the space of possible strategies and to compare the improvements achieved by different strategies. There has to be a variety of different learning times for the individuals instead of the same for all of them, in order to estimate the efficiency of different strategies in every adaptation cycle to find out the putative best strategy. The variable $\tau_i^{(g)} \in \mathbb{N}_0$ denotes the learning time of the i^{th} individual in the g^{th} adaptation cycle.

One possibility for the adaptation would be to understand learning as some kind of operator. For each individual, one operator from a set of learning operators with different learning times is applied and rated depending on the fitness gain. The probability of these operators to be applied in the next adaptation cycle is

adjusted based on the rating (Davis, 1989). In the domain of evolutionary optimization of learning systems, such a procedure was successfully applied by Igel and Kreutz (1999, 2001) for the adaptation of the probabilities to apply different mutation operators. In our context, this procedure is not suitable, because the number of different operators needed to represent the various learning times would be too high. Further, the learning operators strongly differ in their computational costs. To allow for an ongoing probability adaptation, none of these operators should be allowed to become extinct. Hence, even when some expensive operators are not suitable in the current phase of optimization, they have to be applied every now and then and may dominate the average computational complexity. Simulations support this assumption.

Instead of the operator approach, we adapt the learning time more directly. To explore different learning strategies, the value of $\tau_i^{(g)}$ is drawn independently for each individual from a Poisson distribution with the expectation $m^{(g)}$.[1] The parameter $m^{(g)}$ is subject to adaptation, which permits a smooth adjustment of the learning periods. The expectation of the learning time in adaptation cycle $g+1$ is given by

$$m^{(g+1)} = \max\left[(1-\gamma)m^{(g)} + \gamma\,\tilde{\tau}^{(g)}, m_{\min}\right]. \quad (1)$$

The variable $\gamma \in [0,1]$ determines the influence of $\tilde{\tau}^{(g)}$, the value which would have been the most suitable choice of $m^{(g)}$ in the last adaptation cycle. In the end, (1) is a weighted average over the whole history, but the influence of past generations is exponentially suppressed depending on γ. This weighted average is similar to the evolution path in the CMA evolution strategy. The lower bound $m_{\min}$ in (1) enables a minimum diversity of the learning times to allow for continuing adaptation.

Now one is only left with the estimation of $\tilde{\tau}^{(g)}$. The efficiency of learning for τ iterations is measured by the benefit $B^{(g)}(\tau)$ (Tuson and Ross, 1998), normalized to the costs of learning $c(\tau)$ as proposed by Igel and Kreutz (1999):

$$B^{(g)}(\tau) = \frac{1}{N_\tau^{(g)}} \sum_\iota \max\left[\frac{\phi(\text{parent}(\iota)) - \phi(\iota)}{c(\tau)}, 0\right]. \quad (2)$$

The sum runs over all $N_\tau^{(g)}$ offspring ι in adaptation cycle g that have been trained for τ iterations; $\phi(.)$ assigns each individual a fitness value. As an alternative to (2), one might relate $B^{(g)}(\tau)$ only to the fitness gain achieved by learning, i.e., the difference of the offspring's fitness before and after learning replaces the numerator in (2). However, (2) has empirically proven to be more efficient, as it allows for evaluating the number of iterations in the context of mutations. For instance, it is able to take into account the time necessary to counterbalance mutational disturbances.

The computational costs $c(\tau)$ depend on the implementation of the feed-forward neural network. For simplicity, we utilize an approximation. It takes roughly twice as much time to calculate the gradient of the network error with respect to all weights than to compute the network's error itself (Rummelhart et al., 1986): First, the input is "propagated forward" through the network and thereafter it is "propagated backward" through it. Additionally, one "forward-propagation" has to be performed after the last iteration of learning to calculate the individual's fitness $\phi(\iota)$. As we use "propagations" as the unit of the costs, we set

$$c(\tau) = 2\tau + 1 \quad . \quad (3)$$

The learning time $\tilde{\tau}^{(g)}$ should be the time for which the improvements have been maximal in the near past and therefore might also be in the near future. This seems to be fulfilled for $\tilde{\tau}^{(g)} = \arg\left[\max_\tau \left\{B^{(g)}(\tau)\right\}\right]$. However, this might not be optimal, as in the next generations not only learning times equal to $\tilde{\tau}^{(g)}$ are applied, but also learning times randomly drawn from a Poisson distribution. In the limiting case of an isolated maximum of $B^{(g)}(\tilde{\tau}^{(g)})$, learning times $\tau_i^{(g+1)} = \tilde{\tau}^{(g)}$ would yield a maximum improvement, but slightly differing learning times would mainly lead to an exploration of bad strategies. Therefore, we do not consider the maximum of $B^{(g)}(\tau)$, but the maximum of

$$b^{(g)}(\tau) = \sum_{\tau'=0}^{\infty} B^{(g)}(\tau') \cdot \frac{\tau^{\tau'}}{\tau'!}\,\mathrm{e}^{-\tau} \quad , \quad (4)$$

the convolution of $B^{(g)}(\tau)$ with the Poisson distribution with mean τ. The value of $b^{(g)}(\tau)$ is an estimation of the expected improvement in the case of $m^{(g)} = \tau$, as the distribution of learning times is taken into account. As a side effect, the convolution yields a smoothing of $B^{(g)}(\tau)$, which makes the evaluation of the benefit more robust. Finally, the estimation of the optimal learning time is given by

$$\tilde{\tau}^{(g)} = \arg\left[\max_\tau \left[b^{(g)}(\tau)\right]\right] \quad . \quad (5)$$

[1] The Poisson distribution of $\tau_i^{(g)} \in \mathbb{N}_0$ is given by $p_{\tau_i^{(g)}} = \frac{\left(m^{(g)}\right)^{\tau_i^{(g)}}}{\tau_i^{(g)}!}\,\mathrm{e}^{-m^{(g)}}$. As the expectation as well as the variance are equal to $m^{(g)}$, the width of the distribution increases with its expectation.

In (4) the case of non-instantiated learning times, i.e., times that have not been sampled, has to be considered, as for the corresponding values of $B^{(g)}(\tau)$ no estimations are available. This is particularly true for learning periods that strongly deviate from the mean $m^{(g)}$, but by chance it can also occur quite close to $m^{(g)}$. If information about longer and shorter learning times is available, we substitute missing values by means of linear interpolation. Otherwise, we make the most conservative assumption of $B^{(g)}(\tau) = 0$.

One might argue that due to the adaptation of the learning time, some new parameters are introduced and have to be chosen. However, these new parameters are, compared to the learning time, intuitive and therefore much easier to choose. Further, they are not as problem dependent as the fixed learning time parameter is. Therefore, although the absolute number of parameters has increased, the choice of their values has become easier and more robust. In addition, the online adaptation of the learning times adds a new quality to the algorithm, as dissimilar learning times can be realized in different stages of optimization.

3 EXPERIMENTAL VERIFICATION

In the following, we empirically show how the adaptation of the learning times improves the efficiency of the optimization of neural networks. Additionally, we consider the adaptation dynamics of the learning time, which provide insights into the different roles of learning and evolution.

All results presented stem from 50 independently initialized trials per setting. The population size was $\mu = 20$, the tournament size $q = 5$, and the standard deviation for the weight mutations $\sigma = 0.05$. Standard parameters were chosen for the iRprop$^+$ algorithm: $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_0 = 0.05$, $\Delta_{\min} = 10^{-8}$, and $\Delta_{\max} = 50$. The hidden neurons of the neural network had the sigmoidal activation function $f(x) = x/(1 + |x|)$ and the output neurons had linear ones. Adaptation of the learning time was conducted every second generation ($G = 2$) with the adaptation rate $\gamma = 1/4$ and a lower bound of $m_{\min} = 1$. This parameter setting should serve as an example rather than as a recommendation for the optimal choice. Finally, we explore different values for the initial expectation of the learning time $m^{(g)}$ and compare the results with an algorithm with the same value as a constant learning time.

3.1 SAMPLE PROBLEMS

We use established neural network benchmark problems for the formulation of different types of structure optimization tasks, in order to observe different dynamics of the learning time.

3.1.1 Smallest Network for 6-Parity

The task is to find the neural network with the lowest number of weights $n^{(\text{dof})}$ that completely solves the classification problem 6-parity. The network is to decide the parity of a bit-string of length 6 (i.e., whether the number of "1" in this string is even or not). Learning was conducted with all $2^6 = 64$ possible patterns, the target values were given by "0" and "1", and the mean squared error $E^{(\text{mse})}$ was used for training. The individual's fitness consists of three addends, the classification error $E^{(\text{class.})}$, the number of weights of the network, and the mean squared error, evaluated using the whole data set:

$$\phi = \nu E^{(\text{class.})} + n^{(\text{dof})} + E^{(\text{mse})} \quad . \tag{6}$$

The parameter $\nu = 10^6$ was chosen with respect to the expected magnitudes of the different addends to describe the previously mentioned importance of the different optimization goals. Normally, it holds $\nu E^{(\text{class.})} \gg n^{(\text{dof})} \gg E^{(\text{mse})}$. The inclusion of $E^{(\text{mse})}$ yields ongoing optimization, even in the absence of improvements with respect to the main goals. However, reduction of the mean squared error may contribute to better classification performance of the offspring and may prepare the deletion of non-relevant weights.

3.1.2 Prediction of Sunspots

The task is to optimize a neural network with respect to its ability to predict the number of sunspots s_{t+1} in the next year, based on their number in the current and previous years. Figure 1 shows the annual number of sunspots. The states s_t, s_{t-1}, s_{t-3}, and s_{t-7}, each of them normalized to $[0.2, 0.8]$, serve as the input for the network.

In contrast to 6-parity, the fitness is only given by the mean squared error of the network on a particular data set. We distinguish between two different settings:

sunspot$^{(=)}$: The same data points (i.e., the prediction of all years from 1708 to 1938) are used for learning and for fitness evaluation.

sunspot$^{(\neq)}$: The data points used for learning and fitness evaluation are disjoint subsets of the sunspot data set. The prediction error in the years 1708 to

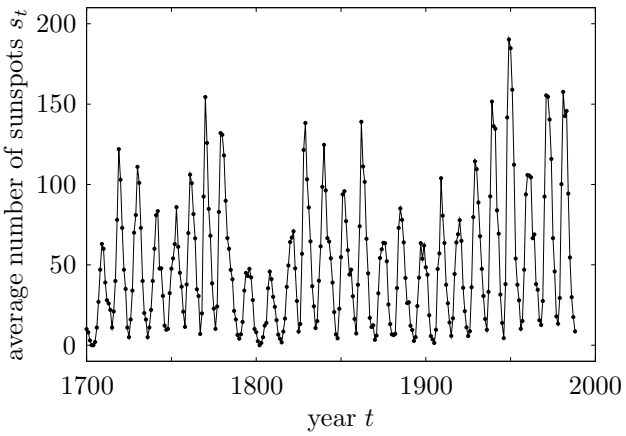


Figure 1: Evolution of the average number of sunspots in the last 300 years. The data are provided by the SIDC (`http://sidc.oma.be`).

1784 is used for learning and the weight configuration of the network with the best error over the years between 1785 and 1861 is inherited (hold-out samples). The individual's fitness is given by the mean squared error in the years between 1862 and 1938.

3.2 DISCUSSION OF RESULTS

The results of the structure optimization for the different problems are summarized in table 1. It becomes obvious that for both algorithms—with and without learning time adaptation—the achieved fitness strongly depends on the initial learning time. In all cases where the differences between both algorithms with equal initial learning time are statistically significant, the algorithm with adaptation performs better than the one with constant learning time; in the other cases we must assume both algorithms to be equally good. In particular, for short initial learning periods the advantage of the adaptation becomes obvious. Here, the adaptation can operate faster and therefore select more suitable strategies, where the speed of adaptation is defined as the change of $m^{(g)}$ per cost unit. The other way round, both algorithms perform equally well for long initial learning times, as the adaptation speed is slow. In 6-parity, after a very high number of generations and for moderate learning times both algorithms have achieved equally good results, as both have "converged". As can be seen from the upper diagram in figure 2, the algorithm with learning time adaptation reaches this state earlier.

Summarizing, learning time adaptation yields better results in most cases and is equally good in the remaining ones. In the following, we consider the learning strategies that emerged for the different problems.

3.2.1 Smallest Network for 6-Parity

The upper diagram in figure 2 shows fitness curves that emphasize the results in table 1. The lower diagram depicts the evolution of the adapted learning times, showing very clear directions of adaptation. The evolution of the learning times and the fitness trajectories in conjunction with (6) allow an identification of three phases of optimization.

1. Initially, the increase of the learning time coincides with a decrease of the classification error.

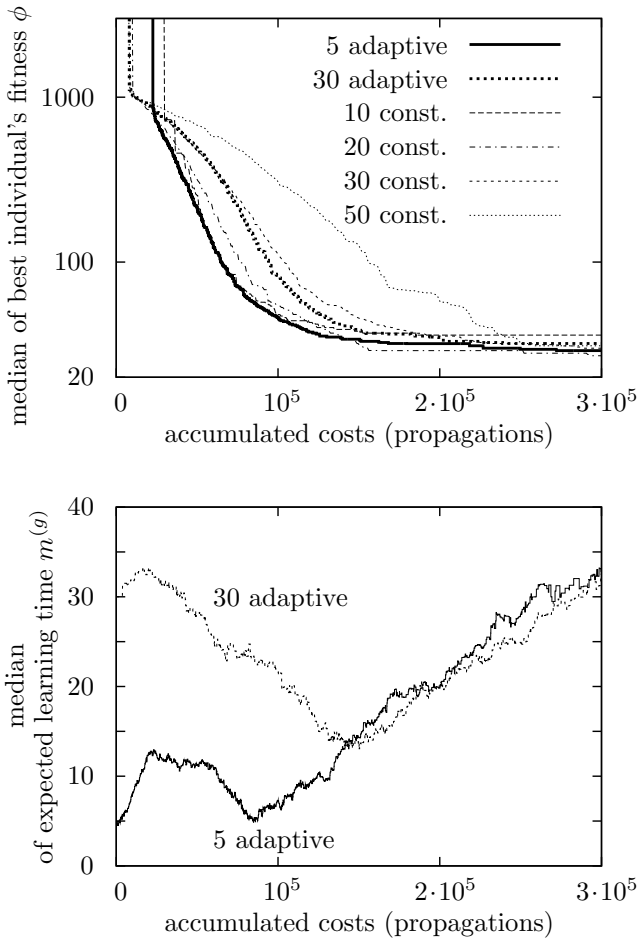


Figure 2: Evolution of the fitness and the adapted learning time for the 6-parity problem, where "adaptive" and "const." refer to the algorithms with and without adaptation of the learning times. The preceding numbers denote either the initial or the constant learning times.

Table 1: Median of the best individual's fitness after a given number of propagations. Depending on the context, the first column shows either the constant learning time or the initial value of the expectation of the learning time. The numbers in parentheses give the number of propagations after which the fitness values were measured. In rows marked with "⋆", the difference between constant and adaptive learning time is statistically significant, Wilcoxon rank sum test (Wilcoxon, 1945), $p < 0.05$.

(initial)	6-parity (10^5)		6-parity ($3{\cdot}10^5$)		sunspot$^{(=)}$ (10^5)		sunspot$^{(\neq)}$ ($2{\cdot}10^5$)	
learning time	constant	adaptive	constant	adaptive	constant	adaptive	constant	adaptive
5	15652.5	46.0 ⋆	15647.1	29.0 ⋆	133.1	84.7 ⋆	139.9	130.3 ⋆
10	46.0	41.5 ⋆	33.0	26.0 ⋆	108.4	83.6 ⋆	144.5	129.2 ⋆
20	59.5	56.0	28.0	28.5	89.7	81.3 ⋆	142.4	128.2 ⋆
30	109.5	83.0 ⋆	30.0	32.0	78.6	74.1 ⋆	137.6	131.8 ⋆
50	323.5	286.5 ⋆	34.0	32.5	74.4	74.2	135.9	138.2

As the networks in the initial population are quite large, it is likely that the algorithm finds suitable architectures within this population to solve the classification problem. Therefore, learning seems to be the driving force in this phase and, in particular for short learning times, the amount of learning is increased compared to the amount of architectural changes.

2. The second phase starts after ϕ has dropped below approximately 1000, i.e., the classification problem is solved and $n^{(\mathrm{dof})}$ becomes the dominating addend in (6). Hence, the amount of the numerator in (2) is dominated by structural changes, i.e., mutations. As the denominator increases with the learning time, it is beneficial to learn only for a short period. In this phase, the task of learning might be to counterbalance mutational "damages" with respect to solving the classification task, rather than to achieve further improvement in the setting of the weights.

3. The third phase is characterized by a continuing increase of the learning time. As a further reduction of $n^{(\mathrm{dof})}$ without worsening the classification performance becomes more and more unlikely, the largest improvement can be gained by reducing the mean squared error. This can efficiently be realized by long learning periods. As the mean squared error is in the order of magnitude of 10^{-4}, these improvements are not visible in figure 2. Reductions of the architecture are rarely observed, maybe prepared by the fine tuning of the weights, as removing of single connections takes place with respect to the corresponding weight (cf. section 2.1).

From this analysis of the three phases, it becomes clear that an algorithm with constant learning time can barely cope with them efficiently, so that learning time adaptation becomes necessary.

3.2.2 Prediction of Sunspots

The results of the experiments using the sunspot data set are shown in figure 3. Interestingly, the evolution of learning times look completely different depending on whether one (sunspot$^{(=)}$) or three different (sunspot$^{(\neq)}$) data sets are involved. In the first case, long learning seems to be a good strategy, as overfitting does not have to be taken into account. A slight reduction of the number of iterations takes place only in the first generations of the trials with an initial learning time of 20, moving the balance towards evolution (see lower left plot in figure 3). This example also shows that the adaptation speed is limited due to the width of the Poisson distribution and the damping in (1) and that the longer the learning time the slower the adaptation.

In sunspot$^{(\neq)}$, where different data sets are used for learning and fitness evaluation, the effect of overfitting becomes important. The lower right plot in figure 3 shows different behaviors depending on the initial learning times. If the adaptive algorithm starts with 5 iterations, the learning time increases during the first generations. This happens because overfitting has not occurred yet and learning is efficient, as improvements on the learning data set coincide with improvements on the fitness data set. However, when after approximately 10^4 propagations overfitting becomes a problem, the learning time strongly decreases and stays at its minimum value $m_{\min}$. In this phase learning would only lead to overfitting and therefore to an increase of the error on the fitness data set, i.e., a worsening of fitness. Progress can only be expected to occur due to fortuitous mutational variations. When the adaptive algorithm is initialized with 20 iterations, the learning

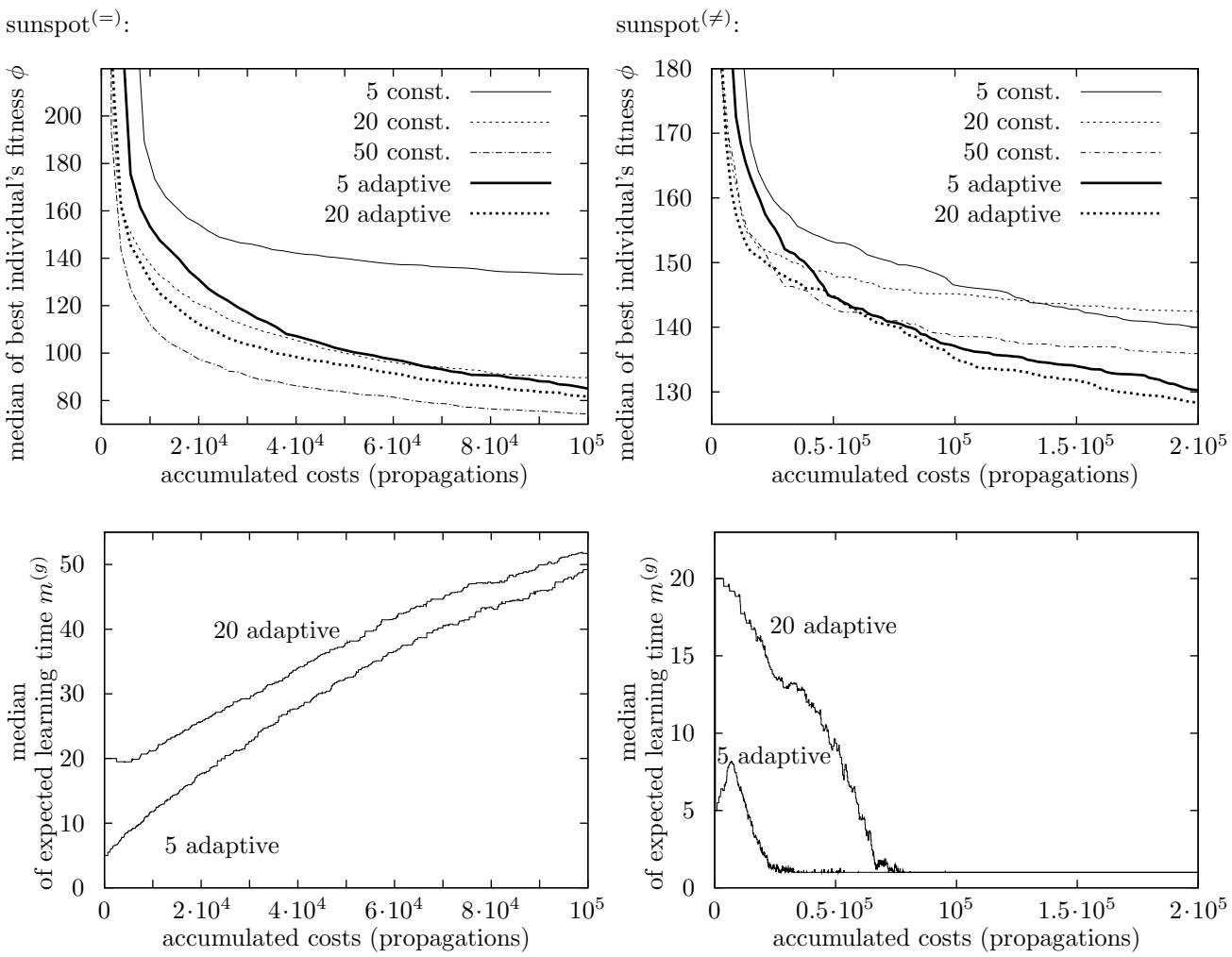


Figure 3: Evolution of the fitness and adapted learning time for the sunspot prediction problem.

time seems to be too long to avoid overfitting right from the start and decreases nearly monotonically to its minimum value.

Again, different learning strategies, as well as a change of the strategy during the course of evolution, can be observed. This explains the significant improvement of the optimization performance by means of adaptive learning time control compared to a constant learning time in all phases of the optimization.

4 CONCLUSION AND OUTLOOK

The number of learning iterations in algorithms that combine learning and evolution is a crucial parameter for the efficiency of the optimization. We have empirically shown that the right learning strategy indeed strongly depends on the problem and on the course of evolution. This may not only include the choice of the initial learning time, but also any schedule for modifying the learning period during optimization. Unfortunately, the best strategy is usually not known *a priori*.

Therefore, we proposed an algorithm that adapts the learning time similarly to strategy parameters of pure evolutionary algorithms. The main idea is to randomize the learning time, to evaluate the fitness improvements resulting from different learning periods, and then to adapt the expectation of the learning time. In a number of examples from the domain of evolutionary optimization of neural networks the adaptation method works in an intuitive way. Moreover, in all examples the optimization is improved due to the better choice of the ratio between evolution and learning.

Based on these findings, our answer to the question of the right choice of the learning time is to start with a small one and use an adaptation scheme to adjust it to the given task. The answer to the question, whether learning is necessary at all, can be left to the evolutionary process itself.

This study is a step towards an evolutionary optimization algorithm for neural networks without any crucial parameters. In further investigations, we plan to combine the control of the learning time with adjustment of the operator probabilities (Igel and Kreutz, 1999, 2001) and adaptation of the population size.

Acknowledgement

We would like to thank the BMBF, grant LOKI, number 01 IB 001 C, for their financial support of our research.

References

Angeline, P. J., G. M. Saunders, and J. B. Pollack (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks 5*(1), 54–65.

Braun, H. (1997). *Neuronale Netze: Optimierung durch Lernen und Evolution*. Springer-Verlag.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 61–69. Morgan Kaufmann.

Eiben, A. E., R. Hinterding, and Z. Michalewicz (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 3*(2), 124–141.

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press.

Hansen, N. and A. Ostermeier (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation 9*(2), 159–195.

Igel, C. and M. Hüsken (2002). Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*. In press.

Igel, C. and M. Kreutz (1999). Using fitness distributions to improve the evolution of learning structures. In *Congress on Evolutionary Computation (CEC '99)*, Volume 3, pp. 1902–1909. IEEE Press.

Igel, C. and M. Kreutz (2001). Operator adaptation in structure optimization of neural networks. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke (Eds.), *Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 1094. Morgan Kaufmann.

Mayley, G. (1996). Landscapes, learning costs and genetic assimilation. *Evolutionary Computation 4*(3), 213–234.

Nolfi, S. and D. Parisi (1999). Learning and evolution. *Autonomous Robots 7*(1), 89–113.

Riedmiller, M. and H. Braun (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pp. 586–591. IEEE Press.

Rummelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning internal representations by error backpropagation. In D. E. Rummelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1, pp. 318–362. MIT Press.

Smith, J. E. and T. C. Fogarty (1997). Operator and parameter adaptation in genetic algorithms. *Soft Computing 1*(2), 81–87.

Tuson, A. and P. Ross (1998). Adapting operator settings in genetic algorithms. *Evolutionary Computation 6*(2), 161–184.

Wilcoxon, F. (1945). Individual comparison by ranking methods. *Biometrics Bulletin 1*, 80–83.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE 87*(9), 1423–1447.

Fuzzy Rule Selection by Data Mining Criteria and Genetic Algorithms

Hisao Ishibuchi
Dept. of Industrial Engineering
Osaka Prefecture University
1-1 Gakuen-cho, Sakai, Osaka 599-8531, JAPAN
E-mail: hisaoi@ie.osakafu-u.ac.jp
Phone: +81-72-254-9350

Takashi Yamamoto
Dept. of Industrial Engineering
Osaka Prefecture University
1-1 Gakuen-cho, Sakai, Osaka, 599-8531, JAPAN
E-mail: yama@ie.osakafu-u.ac.jp
Phone: +81-72-254-9351

Abstract

This paper shows how a small number of fuzzy rules can be selected for designing interpretable fuzzy rule-based classification systems. Our approach consists of two phases: candidate rule generation by data mining criteria and rule selection by genetic algorithms. First a large number of candidate rules are generated and prescreened using two rule evaluation criteria in data mining. Next a small number of fuzzy rules are selected from candidate rules using genetic algorithms. Rule selection is formulated as an optimization problem with three objectives: to maximize the classification accuracy, to minimize the number of selected rules, and to minimize the total rule length. Thus the task of genetic algorithms is to find non-dominated rule sets with respect to the three objectives.

1. INTRODUCTION

Fuzzy rule-based systems have been successfully applied to various fields such as control, modeling, and classification (Leondes 1999). While the main goal in the design of fuzzy rule-based systems has been the performance maximization, their interpretability has also been taken into account in some recent studies (Pene-Reyes & Sipper 1999, Castillo et al. 2001, Roubos & Setnes 2001, and Casillas et al. 2002). In this paper, we consider three objectives in the design of fuzzy rule-based classification systems as in Ishibuchi, Nakashima & Murata (2001): Classification accuracy, the number of fuzzy rules, and the total length of fuzzy rules. The length of a fuzzy rule is the number of its antecedent conditions (i.e., the number of attributes in its antecedent part). The first objective is the performance maximization while the others are related to the interpretability. Usually human users do not want to manually check hundreds of fuzzy rules. Thus the number of fuzzy rules is closely related to the interpretability of fuzzy rule-based systems. Fuzzy rule-based systems with a small number of fuzzy rules are not always interpretable. Human users cannot intuitively understand long fuzzy rules with many antecedent conditions. Thus the rule length is also closely related to the interpretability of fuzzy rule-based systems. In this paper, we maximize the classification accuracy of fuzzy rule-based systems, minimize the number of fuzzy rules, and minimize the total length of fuzzy rules. Multi-objective genetic algorithms are used for finding non-dominated rule sets with respect to these three objectives.

Fuzzy rule generation methods can be categorized into two approaches according to their strategies for dividing the input space into fuzzy subspaces. One approach is based on grid-type fuzzy partitions where the domain interval of each input is divided into antecedent fuzzy sets with linguistic labels. Fig. 1 is an example of such a grid-type fuzzy partition. The other approach uses multi-dimensional antecedent fuzzy sets defined on the input space. Fig. 2 illustrates two-dimensional ellipsoidal antecedent fuzzy sets. Multi-dimensional antecedent fuzzy sets usually lead to fuzzy rule-based systems with high accuracy but low interpretability. On the other hand, fuzzy rule-based systems with high interpretability can be generated from grid-type fuzzy partitions. Since our goal is to generate interpretable fuzzy rule-based systems, we use the first approach (i.e., grid-type fuzzy partitions). As discussed in Suzuki & Furuhashi (2001), homogeneous fuzzy partitions are more interpretable than adjusted ones. Thus we use homogeneous fuzzy partitions as shown in Fig. 1. Usually we do not know an appropriate fuzzy partition for each input. In general, each input may have a different fuzzy partition while the two axes of the input space is divided by the same fuzzy partition in Fig. 1. Moreover, general rules may use coarse fuzzy partitions while specific rules may use fine fuzzy partitions in a single fuzzy rule-based system. For handling such a situation with different fuzzy partitions of different granularities, we specify each antecedent condition of

fuzzy rules by choosing an antecedent fuzzy set from various fuzzy partitions for each input. In this paper, we use four fuzzy partitions in Fig. 3 where the total number of antecedent fuzzy sets is 14. For generating short fuzzy rules with a small number of antecedent conditions, we use "*don't care*" as an additional antecedent fuzzy set. Thus an antecedent fuzzy set for each input is chosen from the 14 fuzzy sets in Fig. 3 and "*don't care*". The total number of combinations of antecedent fuzzy sets is 15^n for an *n*-dimensional pattern classification problem.

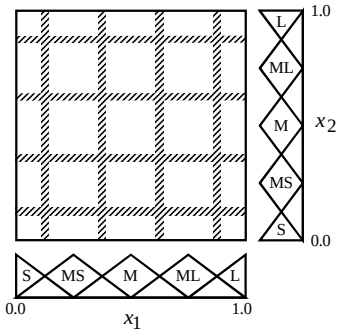


Figure 1: A 5×5 fuzzy grid of a two-dimensional input space.

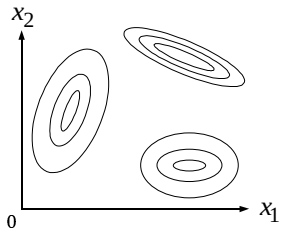


Figure 2: Ellipsoidal antecedent fuzzy sets.

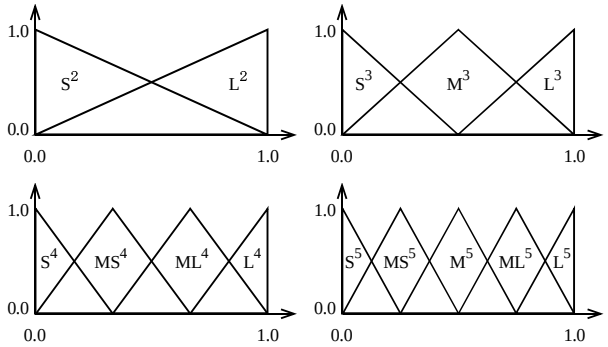


Figure 3: Four fuzzy partitions. The meaning of each label is as follows: S: *small*, MS: *medium small*, M: *medium*, ML: *medium large*, and L: *large*. The superscript of each label denotes the granularity of the corresponding fuzzy partition.

Genetic algorithm-based fuzzy rule selection (Ishibuchi, Nakashima & Murata, 2001) consists of two phases. In the first phase, a large number of candidate rules are generated from various combinations of antecedent fuzzy sets. In the second phase, subsets of the generated candidate rules are examined using genetic algorithms for finding non-dominated rule sets with respect to the above-mentioned three objectives. In Ishibuchi, Nakashima & Murata (2001), a single fuzzy partition was used for all inputs as in Fig. 1. In this case, the total number of combinations of antecedent fuzzy sets including "*don't care*" is $(5+1)^n$ for an *n*-dimensional pattern classification problem. This is much smaller than 15^n in this paper. That is, we have much more candidate rules. It should be noted that the search space for finding non-dominated rule sets exponentially expands as the number of candidate rules increases. The efficiency of genetic algorithms is significantly deteriorated by the increase in the number of candidate rules as shown in this paper. Thus we need a trick for decreasing the number of candidate rules. Our idea is to prescreen candidate rules based on fuzzy versions of two rule evaluation criteria (i.e., *confidence* and *support*) for association rules, which have been frequently used in the field of data mining (Agrawal et al. 1996). In our prescreening procedure, fuzzy rules are divided into several groups according to their consequent classes. Then fuzzy rules in each group are sorted in a descending order of the product of *confidence* and *support*. Finally a pre-specified number of fuzzy rules are chosen from the top of the rule list for each group. The selected fuzzy rules are used as candidate rules in our genetic algorithm-based rule selection method.

In the next section, we show how the design of fuzzy rule-based classification systems can be formulated as a three-objective rule selection problem. In Section 3, we propose a prescreening procedure of candidate rules using fuzzy versions of the two rule evaluation criteria in data mining. In Section 4, we describe a three-objective genetic algorithm for rule selection. The effect of the proposed prescreening procedure on the efficiency of the genetic algorithm-based rule selection method is examined in Section 5 through computer simulations. Finally Section 6 concludes this paper.

2. PROBLEM FORMULATION

Let us consider an *M*-class pattern classification problem with *m* labeled patterns $\boldsymbol{x}_p = (x_{p1}, ..., x_{pn})$, $p = 1, 2,...,m$ in an *n*-dimensional continuous pattern space. For simplicity of explanation, we assume that the pattern space is the *n*-dimensional unit hypercube $[0, 1]^n$. That is, we assume that all attribute values are real numbers in the unit interval $[0, 1]$. For our pattern classification problem, we use fuzzy rules of the following form:

$$\text{Rule } R_q: \text{If } x_1 \text{ is } A_{q1} \text{ and } ... \text{ and } x_n \text{ is } A_{qn} \text{ then Class } C_q \text{ with } CF_q, \quad (1)$$

where R_q is the q-th fuzzy rule, $\boldsymbol{x} = (x_1, ..., x_n)$ is an n-dimensional pattern vector, A_{qi} is an antecedent fuzzy set, C_q is a consequent class (i.e., one of the M classes), and CF_q is a rule weight (i.e., certainty factor). The antecedent fuzzy set A_{qi} is one of the 14 fuzzy sets in Fig. 3 or "*don't care*". The rule weight CF_q is a real number in the unit interval $[0, 1]$. As shown in the next section, the consequent class C_q and the rule weight CF_q are determined in a heuristic manner from compatible training patterns with the antecedent part of R_q.

Let S be a subset of 15^n fuzzy rules of the form (1). Our task is to find rule sets with high classification ability and high interpretability. This task can be rephrased as finding a small number of simple fuzzy rules with high classification ability. As in Ishibuchi, Nakashima & Murata (2001), our rule selection problem is formulated as the following three-objective optimization problem:

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

where $f_1(S)$ is the number of correctly classified training patterns by S, $f_2(S)$ is the number of fuzzy rules in S, and $f_3(S)$ is the total rule length of fuzzy rules in S.

Usually there is no optimal rule set with respect to all the three objectives. Thus our task is to find multiple rule sets that are not dominated by any other rule sets. A rule set S_B is said to dominate another rule set S_A (i.e., S_B is better than S_A : $S_A \prec S_B$) if all the following inequalities hold:

$$f_1(S_A) \le f_1(S_B), \quad (3)$$
$$f_2(S_A) \ge f_2(S_B), \quad (4)$$
$$f_3(S_A) \ge f_3(S_B), \quad (5)$$

and at least one of the following inequalities holds:

$$f_1(S_A) < f_1(S_B), \quad (6)$$
$$f_2(S_A) > f_2(S_B), \quad (7)$$
$$f_3(S_A) > f_3(S_B). \quad (8)$$

The first condition (i.e., all the three inequalities in (3)-(5)) means that no objective of S_B is worse than S_A (i.e., S_B is not worse than S_A). The second condition (i.e., one of the three inequalities in (6)-(8)) means that at least one objective of S_B is better than S_A. When a rule set S is not dominated by any other rule sets, S is said to be a Pareto-optimal solution of our rule selection problem in (2). In many cases, it is impractical to try to find true Pareto-optimal solutions of our rule selection problem whose search space is huge (i.e., the search space is the power set of 15^n fuzzy rules). Thus we try to find near Pareto-optimal solutions. More specifically, first we decrease the search space by prescreening candidate fuzzy rules. Then we search for near Pareto-optimal solutions by a three-objective genetic algorithm.

3. CANDIDATE RULE PRESCREENING

3.1 FUZZIFICATION OF ASSOCIATION RULES

As we have already explained, the total number of combinations of antecedent fuzzy sets is 15^n for our n-dimensional pattern classification problem. When n is small (e.g., $n \le 4$), we can examine all combinations of antecedent fuzzy sets for generating fuzzy rules and use all the generated fuzzy rules as candidate rules in our genetic algorithm-based rule selection method. That is, no prescreening of candidate rules is necessary. On the other hand, we need a prescreening procedure when n is large. It is time-consuming to examine all the 15^n combinations when n is large (e.g., $n = 13$ in wine data used in computer simulations of this paper). In this case, it is also impractical to use all the generated fuzzy rules as candidate rules in our genetic algorithm-based rule selection method. Our idea is to use rule evaluation criteria in data mining for decreasing the number of candidate rules.

In the area of data mining, two criteria called *confidence* and *support* have often been used for evaluating association rules (Agrawal et al. 1996). Our fuzzy rule in (1) can be viewed as an association rule of the form $\boldsymbol{A}_q \Rightarrow C_q$. We use the two criteria for prescreening candidate rules. In this subsection, we show how the definitions of these two criteria can be extended to the case of the fuzzy association rule $\boldsymbol{A}_q \Rightarrow C_q$ (Ishibuchi, Yamamoto & Nakashima, 2001). Similar extensions of the two criteria to fuzzy association rules were also proposed in Hong et al. (2001).

Let D be the set of the given m training patterns $\boldsymbol{x}_p = (x_{p1}, ..., x_{pn})$, $p = 1, 2, ..., m$. The cardinality of D is m (i.e., $|D| = m$). The confidence of $\boldsymbol{A}_q \Rightarrow C_q$ is defined as follows (Agrawal et al. 1996):

$$c(\boldsymbol{A}_q \Rightarrow C_q) = \frac{|D(\boldsymbol{A}_q) \cap D(C_q)|}{|D(\boldsymbol{A}_q)|}, \quad (9)$$

where the denominator $|D(\boldsymbol{A}_q)|$ is the number of training patterns compatible with the antecedent part $\boldsymbol{A}_q$, and the numerator $|D(\boldsymbol{A}_q) \cap D(C_q)|$ is the number of training patterns compatible with both the antecedent part $\boldsymbol{A}_q$ and the consequent class C_q. The confidence c indicates the grade of the validity of $\boldsymbol{A}_q \Rightarrow C_q$. That is, c ($\times 100\%$) of training patterns compatible with $\boldsymbol{A}_q$ are also compatible with C_q. In the case of standard association rules, neither $\boldsymbol{A}_q$ nor C_q is fuzzy. Thus the calculations of $|D(\boldsymbol{A}_q)|$ and $|D(\boldsymbol{A}_q) \cap D(C_q)|$ can be performed by simply counting compatible training patterns. On the other hand, each training pattern has a different compatibility grade $\mu_{\boldsymbol{A}_q}(\boldsymbol{x}_p)$ with the antecedent part $\boldsymbol{A}_q$ when $\boldsymbol{A}_q \Rightarrow C_q$ is a fuzzy association rule. Thus such a compatibility grade should

be taken into account. Since the consequent class C_q is not fuzzy, the confidence in (9) can be rewritten as follows (Ishibuchi, Yamamoto & Nakashima 2001):

$$c(\boldsymbol{A}_q \Rightarrow C_q) = \frac{|D(\boldsymbol{A}_q) \cap D(C_q)|}{|D(\boldsymbol{A}_q)|} = \frac{\sum_{p \in \text{Class } C_q} \mu_{\boldsymbol{A}_q}(\boldsymbol{x}_p)}{\sum_{p=1}^{m} \mu_{\boldsymbol{A}_q}(\boldsymbol{x}_p)} . \quad (10)$$

The compatibility grade $\mu_{\boldsymbol{A}_q}(\boldsymbol{x}_p)$ is usually defined by the product or minimum operator. In this paper, we use the product operator as

$$\mu_{\boldsymbol{A}_q}(\boldsymbol{x}_p) = \mu_{A_{q1}}(x_{p1}) \times \cdots \times \mu_{A_{qn}}(x_{pn}), \quad (11)$$

where $\mu_{A_{qi}}(x_{pi})$ is the membership function of the antecedent fuzzy set A_{qi} (i.e., each triangle in Fig. 3).

On the other hand, the support of $\boldsymbol{A}_q \Rightarrow C_q$ is defined as follows (Agrawal et al. 1996):

$$s(\boldsymbol{A}_q \Rightarrow C_q) = \frac{|D(\boldsymbol{A}_q) \cap D(C_q)|}{|D|} . \quad (12)$$

The support s indicates the grade of the coverage by $\boldsymbol{A}_q \Rightarrow C_q$. That is, s ($\times 100\%$) of all the training patterns are compatible with the association rule $\boldsymbol{A}_q \Rightarrow C_q$ (i.e., compatible with both $\boldsymbol{A}_q$ and C_q). In the same manner as the confidence in (10), the support in (12) can be rewritten as follows (Ishibuchi, Yamamoto & Nakashima 2001):

$$s(\boldsymbol{A}_q \Rightarrow C_q) = \frac{|D(\boldsymbol{A}_q) \cap D(C_q)|}{|D|} = \frac{\sum_{p \in \text{Class } C_q} \mu_{\boldsymbol{A}_q}(\boldsymbol{x}_p)}{m} . \quad (13)$$

3.2 CONSEQUENT CLASS AND RULE WEIGHT

The consequent class C_q of the fuzzy rule R_q with the antecedent part $\boldsymbol{A}_q$ is determined as

$$c(\boldsymbol{A}_q \Rightarrow C_q) = \max\{c(\boldsymbol{A}_q \Rightarrow \text{Class } 1), ..., c(\boldsymbol{A}_q \Rightarrow \text{Class } M)\} . \quad (14)$$

That is, the consequent class has the maximum confidence among the M alternative classes. It should be noted that the same class C_q is obtained for $\boldsymbol{A}_q$ when we use the support s instead of the confidence c. This is because the following relation holds between the confidence c and the support s from their definitions:

$$s(\boldsymbol{A}_q \Rightarrow \text{Class } h) = c(\boldsymbol{A}_q \Rightarrow \text{Class } h) \times \frac{|D(\boldsymbol{A}_q)|}{|D|}, \quad t = 1, 2, ..., M . \quad (15)$$

Since the second term (i.e., $|D(\boldsymbol{A}_q)|/|D|$) of the right-hand side is independent of the consequent class, the class with the maximum confidence is the same as the class with the maximum support. The same class also has the maximum product of these two criteria. Usually we can uniquely specify the consequent class C_q for each combination $\boldsymbol{A}_q$ of antecedent fuzzy sets. Only when multiple classes have the same maximum confidence (including the case of no compatible training pattern with the antecedent part $\boldsymbol{A}_q$: $c(\boldsymbol{A}_q \Rightarrow \text{Class } h) = 0$ for all classes), we cannot specify the consequent class C_q for $\boldsymbol{A}_q$. In this case, we do not generate the corresponding fuzzy rule R_q.

The confidence of R_q can be directly used as its rule weight as in Cordon et al. (1999). Our preliminary simulation results showed that better results were obtained from the following definition of the rule weight than the direct use of the confidence:

$$CF_q = c(\boldsymbol{A}_q \Rightarrow C_q) - c_{\text{Second}}, \quad (16)$$

where c_{Second} is the second largest confidence for the antecedent part $\boldsymbol{A}_q$:

$$c_{\text{Second}} = \max_h \{c(\boldsymbol{A}_q \Rightarrow \text{Class } h) \mid h \neq C_q\} . \quad (17)$$

Our preliminary computer simulations also showed that better results were obtained from the definition in (16) than the following definition used in some studies on fuzzy rule-based classification systems (e.g., Ishibuchi, Yamamoto & Nakashima 2001):

$$CF_q = c(\boldsymbol{A}_q \Rightarrow C_q) - c_{\text{Average}}, \quad (18)$$

where c_{Average} is the average confidence over fuzzy rules with the same antecedent part $\boldsymbol{A}_q$ but different consequent classes:

$$c_{\text{Average}} = \frac{1}{M-1} \sum_{h \neq C_q} c(\boldsymbol{A}_q \Rightarrow \text{Class } h) . \quad (19)$$

3.3 PRESCREENING PROCEDURE

The generated fuzzy rules are divided into M groups according to their consequent classes. Fuzzy rules in each group are sorted in a descending order of the product of the confidence and the support (i.e., $s \cdot c$). For selecting N candidate rules, the first N/M rules are chosen from each of the M groups. In this manner, we can choose a

pre-specified number of candidate rules as candidate rules in our genetic algorithm-based rule selection method. In our preliminary computer simulations, we also examined the confidence and the support as rule prescreening criteria. The best result among the three criteria for rule prescreening (i.e., confidence, support, and their product) was obtained when we used the product of the confidence and the support.

As we have already mentioned, the total number of combinations of antecedent fuzzy sets is 15^n for our *n*-dimensional pattern classification problem. Thus it is impractical to examine all combinations when *n* is large. In this case, we examine only short fuzzy rules with only a few antecedent conditions (i.e., with many *don't care* conditions). The number of fuzzy rules of the length *L* is calculated as ${}_n C_L \times 14^L$ because we have 14 antecedent fuzzy sets for each input (excluding *don't care*). Even when *n* is large, ${}_n C_L \times 14^L$ is not so large for a small value of *L*. This means that the number of short fuzzy rules is not so large even when the total number of fuzzy rules is huge.

4. GENETIC ALGORITHM

Many genetic algorithms for multi-objective optimization problems have been proposed in the literature (Zitzler & Thiele 1999, and Zitzler et al. 2000). Since each rule set can be represented by a binary string, we can apply those algorithms to our three-objective rule selection problem in Section 2. In this paper, we use a slightly modified version of a three-objective genetic algorithm for rule selection in Ishibuchi, Nakashima & Murata (2001). This algorithm has two characteristic features. One is to use a scalar fitness function with variable random weights for evaluating each string (i.e., each rule set). Whenever a pair of parent solutions is selected for crossover, weights are randomly updated. That is, each selection is governed by a different weight vector. Genetic search in various directions in the three-dimensional objective space is realized by this random weighting scheme. The other characteristic feature is to store all non-dominated solutions as a secondary population separately from a current population. The secondary population is updated at every generation. A small number of non-dominated solutions are randomly chosen from the secondary population and their copies are added to the current population as elite solutions. The convergence speed of the current population to Pareto-optimal solutions is improved by the elitist strategy. Other parts of our three-objective genetic algorithm are the same as standard single-objective genetic algorithms. Note that our task is to find multiple non-dominated solutions while the task of standard genetic algorithms is to find a single optimal solution. Of course, we can use other multi-objective genetic algorithms proposed in the literature.

An arbitrary subset S of N candidate fuzzy rules can be represented by a binary string of the length N as

$$S = s_1 s_2 \cdots s_N , \tag{20}$$

where $s_q = 0$ means that the q-th rule R_q is not included in S while $s_q = 1$ means that R_q is included in S. An initial population is constructed by randomly generating a pre-specified number of binary strings of the length N.

The first objective $f_1(S)$ of each string S is calculated by classifying all the given training patterns by S. We use a fuzzy reasoning method based on a single winner rule as in Ishibuchi, Nakashima & Murata (2001). In this fuzzy reasoning method, the classification of each pattern by the rule set S is performed by finding a single winner rule with the maximum product of the rule weight and the compatibility grade with that pattern. There are many cases where some fuzzy rules in S are not chosen as winner rules for any patterns. We can remove those fuzzy rules from S without degrading the classification accuracy of S. At the same time, the second and third objectives are improved by removing unnecessary fuzzy rules. Thus we remove all fuzzy rules that are not selected as winner rules of any patterns from the rule set S. The removal of those rules is performed for each string in the current population by changing the corresponding 1's to 0's before the second and third objectives are calculated.

After the three objectives of each string (i.e., each rule set) in the current population are calculated, the secondary population of non-dominated rule sets is updated. That is, each rule set in the current population is examined whether it is dominated by other rule sets in the current and secondary populations. If it is not dominated by any other rule sets, its copy is added to the secondary population. Then all rule sets dominated by the newly added one are removed from the secondary population. In this manner, the secondary population is updated at every generation.

The fitness value of each rule set S in the current population is defined by the three objectives as

$$fitness(S) = w_1 \cdot f_1(S) - w_2 \cdot f_2(S) - w_3 \cdot f_3(S), \tag{21}$$

where w_1, w_2 and w_3 are weights satisfying the following conditions:

$$w_1, w_2, w_3 \geq 0, \tag{22}$$

$$w_1 + w_2 + w_3 = 1. \tag{23}$$

Whenever a pair of parent strings is selected from the current population, these weights are randomly updated. The random specification of the rule weights is to search for a variety of non-dominated rule sets in the three-dimensional objective space. Binary tournament selection with replacement is used for selecting a pair of parent

strings using the scalar fitness function in (21) with the randomly specified weights. That is, two strings are randomly selected from the current population and the better one is chosen as a parent string. Then the two strings are returned to the current population. The other parent string is also selected in the same manner using the same weight values. When another pair of parent strings is selected, the weight values are randomly updated.

Uniform crossover is applied to each pair of parent strings to generate a new string. Then biased mutation is applied to the generated string for efficiently decreasing the number of fuzzy rules included in the string. In the biased mutation operation, a larger probability is assigned to the mutation from 1 to 0 (i.e., mutation for decreasing the number of fuzzy rules) than the mutation from 0 to 1 (i.e., mutation for increasing the number of fuzzy rules).

The next population consists of the newly generated strings by the genetic operations. Some non-dominated strings in the secondary population are randomly selected as elite solutions and their copies are added to the new population. The outline of the three-objective genetic algorithm for rule selection is written as follows:

Step 0: Parameter Specification.

Specify the population size N_{pop}, the number of elite solutions N_{elite} that are randomly selected from the secondary population and added to the current population, the crossover probability p_c, two mutation probabilities $p_m(1 \to 0)$ and $p_m(0 \to 1)$, and the stopping condition.

Step 1: Initialization.

Randomly generate N_{pop} binary strings of the length N as an initial population. Calculate the three objectives of each string. In this calculation, unnecessary rules are removed from each string. Find non-dominated strings (i.e., non-dominated rule sets) in the initial population. A secondary population consists of copies of those non-dominated strings.

Step 2: Genetic Operations.

Generate $(N_{pop} - N_{elite})$ strings using genetic operations (i.e., binary tournament selection, uniform crossover, and biased mutation) from the current population.

Step 3: Evaluation.

Calculate the three objectives of each of the newly generated $(N_{pop} - N_{elite})$ strings. In this calculation, unnecessary rules are removed from each string. The current population consists of the modified strings.

Step 4: Secondary Population Update.

Update the secondary population by examining each string in the current population as mentioned above.

Step 5: Elitist Strategy.

Randomly select N_{elite} strings from the secondary population and add their copies to the current population.

Step 6: Termination Test.

If the stopping condition is not satisfied, return to Step 2. Otherwise terminate the execution of the algorithm. All the non-dominated strings among examined ones in the execution of the algorithm are stored in the secondary population.

5. COMPUTER SIMULATIONS

We apply the proposed rule selection method to wine data available from the UCI Machine Learning Repository (http://www.ics.uci.edu/~mlearn/MLSummary.html). The wine data set consists of 178 samples with 13 continuous attributes from three classes. We normalized each attribute value into a real number in the unit interval [0, 1]. Thus the wine data set was handled as a three-class pattern classification problem in the 13-dimensional unit hypercube $[0, 1]^{13}$. The total number of possible combinations of antecedent fuzzy sets is 15^{13}.

First we generated fuzzy rules of the length three or less using all the 178 samples as training patterns. The number of generated fuzzy rules of each length is summarized in Table 1. The fuzzy rule of the length zero has no antecedent conditions, Class 2 consequent, and a very small certainty grade (i.e., rule weight). This fuzzy rule can be generated because the number of Class 2 samples is the largest among the three classes in the wine data.

Table 1: The number of generated fuzzy rules of each length.

Length of rules	0	1	2	3	Total
Number of rules	1	182	14,781	696,752	711,716

The generated 711,716 fuzzy rules were divided into three groups according to their consequent classes. Fuzzy rules in each class were sorted in a descending order of the product of the confidence and the support. From each group, the first 300 fuzzy rules were selected as candidate rules ($N = 900$: 900 candidate rules in total). Then the three-objective genetic algorithm was applied to the 900 candidate rules using the following parameter specifications.

Population size: $N_{pop} = 50$,
Number of elite solutions: $N_{elite} = 5$,
Crossover probability: $p_c = 0.9$,
Mutation probability: $p_m(1 \to 0) = 0.1$,
$p_m(0 \to 1) = 1/N$,
Stopping condition: 10,000 generations.

Our computer simulations were iterated 20 times. Non-dominated rule sets obtained from those 20 trials are summarized in Table 2. Examples of the obtained rule

sets in Table 2 are shown in Fig. 4 and Fig. 5. Fig. 4 shows three fuzzy rules with only a single antecedent condition, which correspond to the second rule set with a 94.9% classification rate in Table 2. Fig. 5 shows three fuzzy rules with a few antecedent conditions, which correspond to the sixth rule set with a 100% classification rate in Table 2.

Table 2: Non-dominated rule sets obtained from 20 trials of the proposed method with 900 candidate rules.

Number of rules	Average rule length	Classification rate (%)
3	0.67	88.2
3	1.00	94.9
3	1.33	96.1
3	1.67	98.3
3	2.00	99.4
3	2.33	100.0
4	0.75	96.1
4	1.00	97.2
4	1.25	98.9

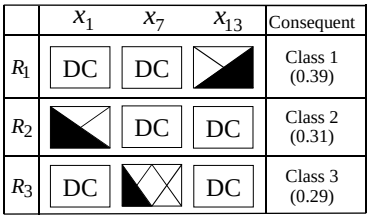


Figure 4: Three fuzzy rules with a 94.9% classification rate.

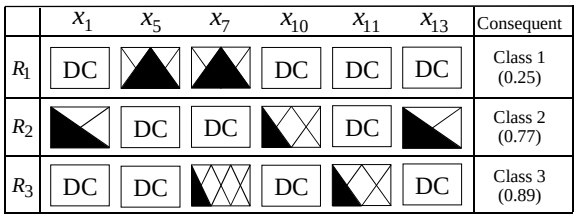


Figure 5: Three fuzzy rules with a 100% classification rate.

From Table 2, we can see that our rule selection method found various rule sets with different classification rates and different sizes. The selected rule sets have high interpretability as shown in Fig. 4 and Fig. 5. From the comparison between Fig. 4 and Fig. 5, we can observe the existence of a tradeoff between classification accuracy and interpretability (i.e., the three fuzzy rules in Fig. 5 have a higher classification rate but less interpretable).

For examining the usefulness of the proposed prescreening procedure of candidate rules, the same computer simulation was performed using randomly selected 900 candidate rules from the generated 711,716 fuzzy rules. Simulation results are summarized in Table 3. From the comparison between Table 2 and Table 3, we can see that the classification ability and/or the interpretability of obtained rule sets were deteriorated by the use of randomly selected candidate rules.

We also performed the same computer simulation using no prescreening procedure. In this case, all the generated 711,716 fuzzy rules were used as candidate rules. Thus the string length was 711,716. As we can expect, the execution of the three-objective genetic algorithm with such a long string required large memory storage and long CPU time. Table 4 shows non-dominated rule sets obtained from ten trials of the three-objective genetic algorithm. Since the search space was too large, good rule sets could not be obtained within a reasonable computation time (especially with respect to the number of fuzzy rules as shown in Table 4). The average CPU time for each trial was about 11 hours in Table 4 while it was about four minutes in Table 2 with 900 candidate rules selected by the proposed prescreening procedure.

Table 3: Simulation results with randomly selected 900 candidate rules.

Number of rules	Average rule length	Classification rate (%)
3	1.67	86.5
3	2.00	93.3
3	2.33	95.5
3	2.67	96.1
4	2.25	96.6
4	2.50	97.2
4	2.75	97.8
5	2.40	98.3
5	2.60	98.9
6	2.50	99.4
7	2.57	100.0
8	2.13	100.0

Table 4: Simulation results with 711,716 candidate rules.

Number of rules	Average rule length	Classification rate (%)
5	1.40	94.4
5	1.60	96.1
6	1.50	96.6
6	1.83	98.3
7	1.71	100.0

Finally we examined the effect of using various fuzzy partitions for each input on the classification performance of fuzzy rule-based classification systems. In the same manner as the computer simulation for Table 2, we applied our rule selection method to the wine data set using only the finest fuzzy partition with five linguistic labels in Fig. 3 (i.e., the bottom-right fuzzy partition in

Fig. 3). Table 5 shows non-dominated rule sets obtained from 20 trials. From the comparison between Table 2 and Table 5, we can see that smaller rule sets with higher classification rates were obtained in Table 2 than Table 5. This result was expected from the fact that the three fuzzy rules with a 100% classification rate in Fig. 5 use various fuzzy partitions with different granularities.

Table 5: Non-dominated rule sets obtained from 20 trials using only a single fuzzy partition with five fuzzy sets.

Number of rules	Average rule length	Classification rate (%)
3	0.67	85.4
3	1.00	91.6
3	1.33	93.3
4	1.00	95.5
4	1.25	96.1
4	1.50	97.2
5	1.00	97.2
5	1.40	97.8
5	1.60	98.3
5	1.80	98.9
6	1.00	97.8
6	1.17	98.3
6	1.33	98.9
6	1.50	99.4
7	1.57	100.0

6. CONCLUSIONS

In this paper, we extended the genetic algorithm-based rule selection method in Ishibuchi, Nakashima & Murata (2001) to the case where various fuzzy partitions with different granularities are used for each input. This extension leads to the increase in the number of candidate rules. Thus we proposed a prescreening procedure for decreasing the number of candidate rules. The proposed prescreening procedure is based on two rule evaluation criteria of association rules in the field of data mining. Through computer simulations, we demonstrated the necessity of candidate rule prescreening in genetic algorithm-based rule selection. The three-objective genetic algorithm could not find good rule sets when candidate rules were randomly chosen. In the case of no prescreening, the CPU time was very long (i.e., about 11 hours) while it was a few minutes in the case with the proposed prescreening procedure.

REFERENCES

R. Agrawal et al. (1996) “Fast discovery of association rules,” in U. M. Fayyad et al. (eds.) *Advances in Knowledge Discovery & Data Mining*, 307-328, AAAI Press, Menlo Park.

J. Casillas, O. Cordón, F. Herrera, and L. Magdalena (2002) *Trade-off between Accuracy and Interpretability in Fuzzy Rule-Based Modeling*, Physica-Verlag.

L. Castillo, A. Gonzalez, and P. Perez (2001) “Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm,” *Fuzzy Sets and Systems* **120**, 309-321.

O. Cordon, M. J. del Jesus, and F. Herrera (1999) “A proposal on reasoning methods in fuzzy rule-based classification systems,” *International Journal of Approximate Reasoning* **20**, 21-45.

U. M. Fayyad and K. B. Irani (1993) “Multi-interval discretization of continuous-valued attributes for classification learning,” *Proc. of 13th International Joint Conference on Artificial Intelligence*, 1022-1027.

T. -P. Hong, C. -S. Kuo, and S. -C. Chi (2001) “Trade-off between computation time and number of rules for fuzzy mining from quantitative data,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **9**, 587-604.

H. Ishibuchi, T. Nakashima, and T. Murata (2001) “Three-objective genetics-based machine learning for linguistic rule extraction,” *Information Sciences* **136**, 109-133.

H. Ishibuchi, T. Yamamoto, and T. Nakashima (2001) “Fuzzy data mining: Effect of fuzzy discretization,” *Proc. of 1st IEEE International Conference on Data Mining*, 241-248.

C. T. Leondes (1999) *Fuzzy Theory Systems: Techniques and Applications (Vols. 1-4)*, Academic Press, San Diego.

C. A. Pene-Reyes and M. Sipper (1999) “Designing brest cancer diagnostic systems via a hybrid fuzzy-genetic methodology,” *Proc. of IEEE International Conference on Fuzzy Systems* **1**, 135-139.

J. R. Quinlan (1993) *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo.

H. Roubos and M Setnes (2001) “Compact and transparent fuzzy models and classifiers through iterative complexity reduction,” *IEEE Trans. on Fuzzy Systems* **9**, 516-524.

T. Suzuki and T. Furuhashi (2001) “Evolutionary algorithm based fuzzy modeling using conciseness measure,” *Proc. of Joint IFSA-NAFIPS International Conference*, 1575-1580.

E. Zitzler, K. Deb, and L. Thiele (2000) “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results,” *Evolutionary Computation* **8**, 173-195.

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

Neuron Reordering for Better Neuro-Genetic Hybrids

Jung-Hwan Kim and Byung-Ro Moon
School of Computer Science and Engineering
Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{aram,moon}@soar.snu.ac.kr

Abstract

We propose an approach to improve the performance of neuro-genetic hybrids. We used a two-dimensional genetic encoding to represent the network with less information loss and devised a careful restructuring of neural network so that the geographic contributions of neurons can be better reflected in the genetic process. We test handwritten character recognition problem to examine the performance of the proposed approach. Experimental results showed significant improvement by neuron reordering and two-dimensional encoding.

1 Introduction

Artificial neural networks (ANNs) have been intensively studied for Handwritten Character Recognition (HCR) systems [2, 18, 19, 25]. There are a variety of tuning methods for ANNs. Generally *backpropagation* is the most popular local gradient training technique. However, there is no guarantee with few exceptions that such methods find an optimal neural network. They tend to become stuck at local optima.

We propose a neuro-genetic hybrid approach for improving the performance of neural networks. Genetic algorithms (GAs) can provide diverse initial solutions to backpropagation and have recently become popular [9, 14, 17, 23]. To represent solutions, GAs for ANN optimization have been using linear encodings following the convention of the GA community [10, 14, 20]. Typically, every weight in an ANN takes a position in a linear chromosome of a GA. However, linear encodings have limited capability in reflecting the geographic linkages of genes [1, 7] and a number of two-dimensional (2D) encodings have been used for other problems [1, 4, 7, 15]. The weights of an ANN can be represented by a 2D matrix and thus they are intrinsically suitable for a 2D encoding.

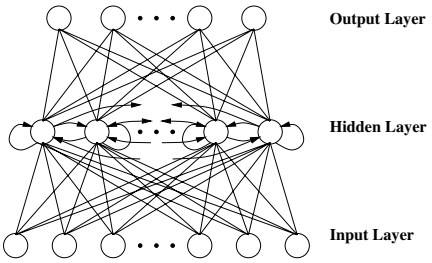


Figure 1: The recurrent neural network architecture we used

In this paper, we use a 2D encoding for genetic representation and employ 2D *geographic crossover* which has demonstrated good performance for the graph partitioning problem [15, 21, 22]. Once the structure of an ANN is fixed, the genotype depends on the order of neurons in the structure. Even with a 2D encoding, the geographic relationships between neurons may not be well reflected. We measure the relative relationships of neurons and propose a restructuring algorithm based on these relative relationships. This restructuring approach enables the hidden neurons with the relatively high relationship to be clustered.

The remainder of this paper is organized as follows. In the next section, we summarize the neural network architecture that we used. In Section 3, we describe our hybrid neuro-genetic approach and the reordering approach. We present our experimental results in Section 4 and state our conclusions in Section 5.

2 Neural Network Design

We use a recurrent neural network architecture based on Elman's recurrent neural network [8]. It consists

of two layers, excluding the input layer, as shown in Figure 1.

Each hidden unit is connected to itself and also fully connected to all the other units. These connections are updated in the propagation phase every Δt time interval. The intermediate output (ϕ_j) of the j^{th} hidden unit at cycle t is computed through the summation of two factors as in the following:

$$\begin{aligned} \phi_j(t) &= \sum_{i=1}^{n} w_{ji} y_i(t) + T \cdot \sum_{i=1}^{p} u_{ji} v_i(t-1) \\ v_j(t) &= \varphi(\phi_j(t)) \end{aligned}$$

where

- $v_j(t)$: the output value of the j^{th} hidden unit at cycle t
- $y_i(t)$: the value of the i^{th} input unit at cycle t
- w_{ji} : the weight between the j^{th} hidden unit and the i^{th} input unit
- u_{ji} : the recurrent weight between the j^{th} hidden unit and the i^{th} hidden unit
- n, p : the numbers of input units and hidden units, respectively
- T : the decay factor to control the reflection ratio of the recurrent factor
- φ : sigmoidal transfer function $\frac{1}{1+e^{-\lambda x}}$.

The first term is related to the feedforward connections between the input and hidden layer and the second term reflects the recurrent factor.

We represent the set of weights by a matrix $M = [m_{ij}]$ where $m_{ji} = w_{ji}$ for $1 \le i \le n$, and $m_{ji} = u_{j,i-n}$ for $n < i \le n+p$. We will create a chromosome based on this matrix. In the m_{ji}, i is the index of the source unit and j is the index of the target unit. Then, in the weight matrix, values on the same row are with the same target unit and values on the same column are with the same source unit.

The weight correction of the above recurrent neural network during error backpropagation is determined as follows:

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \sum_{k=1}^{q} \{-(t_k - o_k)\varphi'(\psi_k)\pi_{kj}\}\,\varphi'(\phi_j) y_i \\ \frac{\partial E}{\partial u_{ji}} &= T \sum_{k=1}^{q} \{-(t_k - o_k)\varphi'(\psi_k)\pi_{kj}\}\,\varphi'(\phi_j) v_i \end{aligned}$$

where

- E : the error value in the training phase calculated using the mean square error
- t_k : the target output of the k^{th} output unit
- o_k : the real output of the k^{th} output unit
- π_{kj} : the weight between the k^{th} output unit and the j^{th} hidden unit
- q : the number of output units
- $\psi_k = \sum_{j=1}^{p} \pi_{kj} v_j$.

We use 28×28 images of digits as input data of the neural network. If the network has an input unit for every one of the $28 \times 28 = 784$ pixels, the problem space becomes very large and the backpropagation learning speed is considerably reduced. Kohara and Nakamura [16] suggested an alternative approach that counts the density (number) of black pixels on each row and provides the vector of the counted numbers. The horizontal density distributions (HDDs) – the spectrum of black-pixel numbers in the horizontal rows – are visibly different digit by digit. This approach can significantly decrease the number of units in the neural networks. They reported an 85.3% recognition rate for the handwritten digit recognition. When this approach is applied to our problem, the number of input units reduces from 784 to 28. This speeds up the convergence of network tuning at the cost of lower recognition quality. We find a compromise in between the two extremes. In addition to the number of black pixels in each row, we also count the number of black pixels in each column (we call this the vertical density distribution (VDD)); thus we have 56 input units. In addition, we shift every image upwards and to the left so that both the leftmost column and the topmost row have at least one black pixel. All our training patterns were fitted to a 23×22 grid. Thus the number of input units becomes 45. We used 20 units in the hidden layer; the output layer naturally has 10 units corresponding to the 10 digits. If any test pattern does not fit to the 23×22 grid, the rightmost columns and bottommost rows are thrown away. We use the *1-of-N* approach as output encoding. Each output neuron corresponds to one of the 10 digits. For each input vector, only one of 10 output neurons has value 1 and the others have value 0.

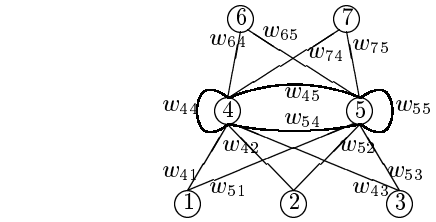
3 The Proposed Neuro-Genetic Hybrid

3.1 The GA Structure

The template of the proposed genetic algorithm is shown in Figure 2. It is a typical steady-state hybrid genetic algorithm. When a genetic algorithm is hybridized with a local improvement heuristic, it called

```
Create initial population of fixed size;
do {
    choose parent1 and parent2 from population;
    offspring = g2d_xover(parent1, parent2);
    mutation(offspring);
    backpropagation(offspring);
    reordering(offspring);
    if suited(offspring)
    then replace(population, offspring);
} until (stopping condition);
Return the best solution;
```

Figure 2: The template of the hybrid GA



(a) NN structure

	input units			hidden units		output units	
	1	2	3	4	5	6	7
4	w_{41}	w_{42}	w_{43}	w_{44}	w_{45}	w_{64}	w_{74}
5	w_{51}	w_{52}	w_{53}	w_{54}	w_{55}	w_{65}	w_{75}

(b) Weight Matrix

Figure 3: A 2D encoding for a neural network.

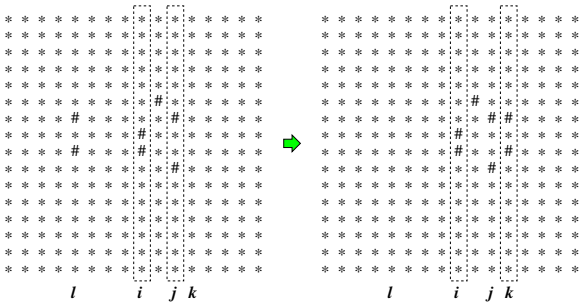


Figure 4: Examples of 2D schemata

a hybrid GA. We set the population size to be 50. Two parents are selected according to probabilities that are proportional to their fitness values. The probability that the best solution is chosen was given as four times that of the worst solution. The offspring is produced through geographic 2D crossover. The "g2d_xover" indicates the geographic 2D crossover in Figure 2. The geographic 2D crossover is described in Section 3.4. The offspring is then modified by a *mutation* operator and locally optimized by backpropagation. The backpropagation process helps the GA fine-tune around local optima. From another perspective, the GA provides diverse initial solutions to the backpropagation routine. At this point, the offspring is modified by reordering the hidden units. Of course, the modification does not affect the quality of the offspring. Then the offspring replaces a solution in the population by the following rule [5]: the more similar parent to the offspring is replaced if the offspring is better; otherwise, the other parent is replaced if the offspring is better; if not again, the worst chromosome in the population is replaced. The rational behind this is to maintain the population diversity to the extent that not too much time is wasted. For stopping, we use the fixed generation.

3.2 Problem Encoding

Most GAs for neural-network optimization encode a solution (a set of weights) with a linear string following the convention [10, 14, 20]. However, linear encodings are known to be a factor limiting GAs' performance in other problems [1, 7]. Two-dimensional encodings have been proven to perform favorably [1, 4, 7, 21, 22]. Instead of transforming into a linear string, we represent a solution by a weight matrix. In the matrix, each row corresponds to a hidden unit and each column corresponds to an input unit, a hidden unit, or an output unit. Figure 3 shows an example of such encoding. The relationships among the edges in the network can be better reflected in this 2D representation, which will be experimentally supported in Section 4. In addition, we suspect that some units have stronger relationships with one another than with the others. We further modify the chromosome by reordering the units according to their relative relationships (See Section 3.3). In summary, a chromosome is represented by a $p \times (p+n+q)$ matrix with columns and rows reordered, where p, n, and q are the numbers of hidden units, input units, and output units, respectively.

3.3 Neuron Reordering (Network Restructuring)

3.3.1 Schemata

A schema is a similarity template describing a subset of strings with similarities at certain string positions inside chromosomes [13]. In a linear encoding of m genes, a schema is defined to be an m-tuple $\langle s_1 s_2 \cdots s_m \rangle$ where $s_i \in U \cup \{*\}$, given a set of alphabet U. In a schema, the symbol '$*$' represents "don't-care" locations and the other symbols represent *specific symbols* which specify the pattern. In the study, a chromo-

```
Calculate d_ij (i, j = 1, 2, ..., p);
Choose the pair (u_m, u_n) (m ≠ n) s.t. d_mn is maximum over all d_ij's;
S = u_m u_n;
U = {u_1, u_2, ..., u_p} − {u_m, u_n};
while (U ≠ φ ) {
    Choose u_l ∈ U s.t. L(u_l, S) is maximum over all L(u_i, S)'s;
    Choose u_r ∈ U s.t. R(u_r, S) is maximum over all R(u_i, S)'s;
    if (L(u_l, S) < R(u_r, S)) {
        S = S · u_r;            // concatenation
        U = U − {u_r};
    } else {
        S = u_l · S;            // concatenation
        U = U − {u_l};
    }
}
```

Figure 5: Reordering algorithm

some is a two-dimensional matrix of gene values. Here, a schema can be represented by a $p \times (p+n+q)$ matrix S where $s_{ij} \in U \cup \{*\}$, $U = [-0.5, 0.5]$. Although a GA handles the chromosomes, it also implicitly handles schemata. From one point of view, the process of a GA is a struggle among schemata. Crossover emergently creates new larger schemata from smaller schemata. To construct a larger schema, corresponding smaller schemata must be preserved through the crossover. In the case of linear chromosomes, schemata with short defining lengths[1] or those with clustered specific-symbol distribution turned out to be advantageous in survival [6, 13]. It is intuitively clear that, in 2D chromosomes too, schemata with clustered specific-symbol distributions would have an advantage. Figure 4 shows two examples of 2D schemata. For convenience, we use '#' to represent the positions of specific symbols. In the left-hand schema, the specific symbols on the column l depart from the specific symbols between columns i and j. If the columns l and k are swapped, the specific symbols are clustered and the schema would be better preserved through crossover. This swap corresponds to a swap between two hidden nodes in the neural network structure. This is done in the process of reordering which is described in the next section.

3.3.2 Reordering

In the hidden layer, a unit inhibits or activates other units. We believe that edges connecting units with strong relationships would have stronger patterns than a random set of edges. If units with a high relative contribution can stay close to one another in the chromosome, schemata related to those units would highly probably survive better through crossover. The larger the weight from unit u_i to u_j, the more strongly unit u_i activates unit u_j. On the other hand, smaller negative weight means stronger inhibition. If the unit u_i strongly activates or inhibits unit u_j, we consider unit u_i to have a high relative contribution to unit u_j.

The reordering algorithm is shown in Figure 5. In the algorithm, the function R, L computes the relative contribution of unit u_i on the string $S = u_a u_{a+1} \cdots u_b$ as follows:

$$\begin{aligned} R(u_i, S) &= \alpha \cdot d_{i,b} + (1-\alpha) \cdot d_{i,b-1} \\ L(u_i, S) &= \alpha \cdot d_{i,a} + (1-\alpha) \cdot d_{i,a+1} \end{aligned}$$

where

$$\begin{aligned} &d_{ij} = z_{ij} + z_{ji}, \\ &z_{ij} = \ \mid (w_{ij} - m_i)/\sigma_i \mid, \\ &w_{ij} : \text{the weight from hidden unit } u_j \text{ to } u_i, \\ &m_i = (\textstyle\sum_{j=1}^{p} w_{ij})/p, \text{ and} \\ &\sigma_i = \sqrt{(\textstyle\sum_{j=1}^{p} (w_{ij} - m_i)^2)/p}\ . \end{aligned}$$

In the expression, d_{ij} is the relative contribution between u_i and u_j. The reordering algorithm helps units with high relative contribution stay close together in the array of neurons. This helps their relevant genes (weights) stay close together in the 2D chromosomes, too. Figure 6 shows an example of such unit reordering. By neuron reordering, schemata are transformed to different ones. The schema in Figure 6 was transformed to a seemingly better one to survive. The effectiveness of the reordering will be examined in Section 4.

[1] The length between the leftmost specific symbol and the rightmost specific symbol.

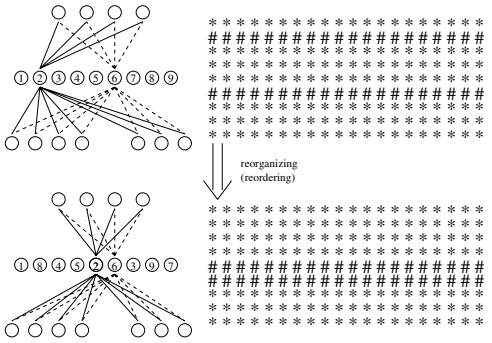


Figure 6: Example of unit reordering

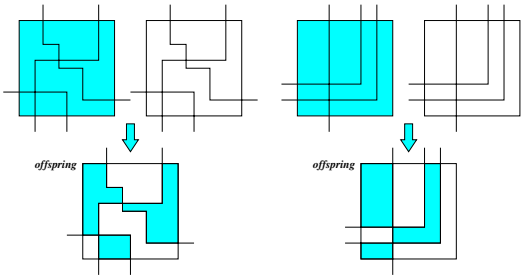


Figure 7: Example of 2D geographic crossover

Bui and Moon [3] proposed a static reordering technique where the genes are reordered just once before the GA starts. The reordering here is different from [3] in that it reorders the neurons every time an offspring is created.

3.4 Geographic 2D Crossover

Two-dimensional encoding can preserve more geographical relationships among the genes [4, 15]. However, when traditional straight-line-based cutting strategies are used, the power of new-schema creation is far below that of crossovers on linear encodings [15]. Geographic crossover was suggested to resolve this problem [15]. In the case of a 2D encoding, it chooses a number of lines, divides the chromosomal domain into two equivalence classes, and alternately copies the genes from the two parent chromosomes. Figure 7 shows two example geographic crossover operators. We used geographic crossover in this work. By combining two-dimensional representation, unit reordering, and 2D geographic crossover, we are pursuing both reduced information loss in the stage of encoding and the power of new-schema creation.

Figure 8: Sample input data

4 Experimental Results

In this section, we examine the performance of the proposed hybrid neuro-genetic algorithm. We used the MNIST database[2] for input patterns of 28×28 grids. We used 800 samples for training, 200 samples for the validation set, and 2000 samples for the test set. The samples are evenly classified to ten digits. Figure 8 shows some samples. The digit images were written by 500 different writers. Simard *et al.* [24] experimented with the same data and used a 3-layer neural network. In their experiment, the recognition rate of the test data was 97.05%. However, they used one input unit for each pixel (i.e., $28 \times 28 = 784$ input units in total) and 650 hidden units. We wish to obtain comparable results to those of [24] using a much simpler model. We used 45 input units and 20 hidden units.[3]

First, we examine the recognition power of the method with HDD of the input images. The weights are tuned by backpropagation only. The initial weights of the network are initialized at random between -0.5 and 0.5. The learning rate is 0.8 and the momentum factor is used. All the other experiments in this paper are also based on this scenario. This method with only HDD input showed 86.00% recognition rate. This recognition rate is consistent with the HDD model of Kohara and Nakamura's (85.3%) [16]. Also, we examine the recognition power of the method with both HDD and VDD information. The recognition power was notably better than the HDD-only model. The addition of VDD information was realized at negligible cost. We use the HDD+VDD model in the hybrid GAs.

Next, we examine the performance of the neuro-genetic hybrids. We tested four versions which are classified according to the utilization of reordering and

[2] http://www.research.att.com/~yann/exdb/mnist

[3] Since we used a recurrent model, the number of edges is greater than the feed-forward model with the same number of units. However, the total number of edges is 1,295, which is much smaller than the 516,100 edges of Simard *et al.*'s study.

Table 1: Results of Neuro-Genetic Hybrid with Given Ordering

	0	1	2	3	4	5	6	7	8	9
	Linear Xover (94.55%)									
0	**192**	2	0	0	0	2	0	0	1	3
1	1	**194**	0	0	0	2	3	0	0	0
2	0	0	**194**	0	0	1	1	3	1	0
3	0	0	2	**185**	0	1	0	0	12	0
4	0	0	0	4	**186**	1	0	1	4	4
5	2	3	0	1	4	**188**	0	0	2	0
6	2	2	0	0	0	4	**192**	0	0	0
7	0	0	0	0	0	2	2	**189**	6	1
8	0	3	1	5	1	0	0	2	**187**	1
9	4	0	0	1	3	2	0	2	4	**184**
	Geographic 2D Xover (95.20%)									
0	**190**	0	0	0	0	3	0	0	1	6
1	0	**195**	0	0	0	2	2	1	0	0
2	1	4	**193**	0	0	0	1	1	0	0
3	0	2	9	**183**	0	0	0	0	6	0
4	0	0	0	3	**186**	0	0	1	1	9
5	4	0	0	0	4	**189**	0	0	1	2
6	1	1	0	0	1	2	**195**	0	0	0
7	0	0	1	0	0	0	3	**195**	0	1
8	4	0	0	3	0	0	0	1	**191**	1
9	4	0	0	1	3	0	0	1	4	**187**

Table 2: Results of Neuro-Genetic Hybrid with Reordering

	0	1	2	3	4	5	6	7	8	9
	Linear Xover (93.85%)									
0	**187**	0	0	0	0	8	2	0	1	2
1	0	**193**	2	0	0	0	1	3	1	0
2	0	4	**192**	1	1	0	0	1	1	0
3	0	0	8	**183**	1	1	0	2	5	0
4	1	0	0	0	**185**	0	0	0	0	14
5	6	2	0	0	1	**188**	3	0	0	0
6	0	3	0	0	1	2	**194**	0	0	0
7	0	1	1	1	0	1	2	**190**	4	0
8	1	0	0	6	1	4	0	3	**183**	2
9	3	2	0	0	7	2	0	2	2	**182**
	Geographic 2D Xover (97.10%)									
0	**193**	0	0	0	0	5	1	0	0	1
1	0	**196**	0	0	0	0	4	0	0	0
2	0	1	**193**	0	1	0	2	2	1	0
3	0	0	2	**196**	1	0	0	0	1	0
4	1	0	0	0	**196**	0	0	0	0	3
5	4	0	0	0	3	**191**	2	0	0	0
6	0	1	0	0	1	1	**196**	0	1	0
7	0	0	1	1	0	0	2	**196**	0	0
8	0	0	1	4	0	1	0	1	**193**	0
9	2	0	0	1	3	0	1	0	1	**192**

2D geographic crossover. The framework of the neuro-genetic hybrids was described in Figure 2. In this framework, the backpropagation adopted the stopped learning method [11, 12].

Table 1 represents the results without reordering. The first group represents the recognition results with the traditional multi-point crossover under the linear encoding. The second group represents the results with the geographic crossover under the 2D encoding. They were both superior to the neural network with only backpropagation. The 2D encoding showed slight improvement over the linear encoding. Table 2 represents the results with reordering. The effectiveness of the 2D encoding and the 2D geographic crossover was notable.

Table 3 shows a summary of the experimental results. The mean and best results are derived from 500 trials for each version. BP1 represents the experimental results with HDD (Kohara – Nakamura version). BP2 indicates the version with HDD + VDD. BP1 and BP2 used only backpropagation. The versions GA1 through GA4 are hybrid GAs with backpropagation and correspond to Table 1 and Table 2. The effectiveness of the geographic 2D crossover was consistent in both non-reordered and reordered cases. The reordering significantly improved the results with the 2D encoding. However, it was not helpful with the linear encoding. This phenomenon appears to occur because the reordering is performed not with the edges but with the neurons (hidden nodes); the order of columns and rows in the 2D encoding are determined by the reordering of neurons. The combination of the 2D crossover and reordering showed strong synergy. Note

Table 3: Summarization of the Result over 500 Trials

	Xover	Reordering	Mean (%)	Best (%)
BP1	—	—	85.35	86.00
BP2	—	—	90.28	93.45
GA1	1D	N	93.38	94.55
GA2	2D	N	94.08	95.20
GA3	1D	Y	93.49	93.85
GA4	2D	Y	96.52	97.10

that the recognition rate 97.10% of GA4 (with 75 units and 1,295 edges in total) is comparable to the value of 97.05% of Simard *et al.*'s study [24] which used a far more complex structure (with 1,444 units and 516,100 edges in total).

5 Concluding Remarks

We proposed a neuro-genetic hybrid approach for handwritten character recognition. To reduce the size of ANNs, we used two vectors representing horizontal density distributions and vertical density distributions for input. This model provides a dimensional reduction of the problem and its actual performance was comparable to the one-unit-per-pixel model.

We devised a reordering algorithm of neurons to effectively reflect the neurons' geographic relationship in the genetic search. We also used a 2D encoding and 2D geographic crossover. The 2D encoding/crossover and reordering showed strong synergy. It is notable that we used an ANN model with 1,295 edges and obtained results that are comparable to those achieved with a huge model with 516,100 edges.

The ideas we have presented here are applicable not just to handwritten character recognition. Future studies will include extending the experiments to other problems.

Acknowledgments

This research was supported in part by KOSEF through Statistical Research Center for Complex Systems at Seoul National University and Brain Korea 21 Project. The RIACT at Seoul National University provided research facilities for this study.

References

[1] C. A. Anderson, K. F. Jones, and J. Ryan. A two-dimensional genetic algorithm for the Ising problem. *Complex Systems*, 5:327–333, 1991.

[2] S. Behnke, M. Pfister, and R. Rojas. Recognition of handwritten digit using structural information. In *International Conference on Neural Networks (ICNN'97)*, volume 3, pages 1391–1396, 1997.

[3] T. N. Bui and B. R. Moon. Hyperplane synthesis for genetic algorithms. In *International Conference on Genetic Algorithms*, pages 102–109, 1993.

[4] T. N. Bui and B. R. Moon. On multi-dimensional encoding/crossover. In *International Conference on Genetic Algorithms*, pages 49–55, 1995.

[5] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.

[6] T. N. Bui and B. R. Moon. GRCA: A hybrid genetic algorithm for circuit ratio-cut partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(3):193–204, 1998.

[7] J. P. Cohoon and W. Paris. Genetic placement. In *IEEE International Conference on Computer-Aided Design*, pages 422–425, 1986.

[8] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[9] A. Grauel and F. Berk. Mapping of dynamical systems by recurrent neural networks in an evolutionary algroithm approach. In *European Congress on Intelligent Techniques and Soft Computing*, volume 1, pages 470–476, 1998.

[10] S. A. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural networks. In *International Conference on Genetic Algorithms*, pages 360–369, 1989.

[11] M. H. Hassoun, P. B. Watta, and R. Shringarpure. Cross-validation without a validation set in bp-trained neural nets. In *IEEE International Conference Neural Networks*, volume 1, pages 369–372, 1995.

[12] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Prentice Hall, 1999.

[13] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, 1975.

[14] R. Jeff and V. B. Ciesielski. An evolutionary approach to training feed-forward and recurrent neural networks. In *International Conference on Knowledge-Based Intelligent Electronics Systems*, pages 596–602, 1998.

[15] A. B. Kahng and B. R. Moon. Toward more powerful recombinations. In *International Conference on Genetic Algorithms*, pages 96–103, 1995.

[16] K. Kohara and Y. Nakamura. Modifying desired outputs to improve pattern recognition by combining subfeature-input neural networks. *Transactions of the Institute of Electrical Engineers of Japan*, 117-c(6):805–813, 1997.

[17] T. Kumagai, M. Wada, S. Mikami, and R. Hashimoto. Structured learning in recurrent neural network using genetic algorithm with internal copy operator. In *IEEE International Intermag '97 Magnetics Conference*, pages 651–656, 1997.

[18] S. J. Lee and H. L. Tsai. Pattern fusion in feature recognition neural networks for handwritten character recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 28(4):612–617, 1998.

[19] S. W. Lee and H. H. Song. A new recurrent neural-network architecture for visual pattern recognition. *IEEE Transactions on Neural Networks*, 8(2):331–340, 1997.

[20] C. T. Lin and C. P. Jou. Controlling chaos by GA-based reinforcement learning neural network. *IEEE Transactions on Neural Networks*, 10(4):846–869, 1999.

[21] B. R. Moon and H. N. Kim. Effective genetic encoding with a two-dimensional embedding

heuristic. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 3(2):113–120, 1999.

[22] B. R. Moon, Y. S. Lee, and C. K. Kim. GEORG: VLSI circuit partitioner with a new genetic algorithm framework. *Journal of Intelligent Manufacturing*, 9(5):401–412, 1998.

[23] V. Patridis, E. Paterakis, and A. Kehagias. A hybrid neural-genetic multimodel parameter estimation algorithm. *IEEE Transactions on Neural Networks*, 9(5):862–876, 1998.

[24] P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognition, tangent distance and tangent propagation. In G. Orr and K. Muller, editors, *Neural Networks: Tricks of the Trade*. Springer, 1998.

[25] B. Zhang, M. Fu, H. Yan, and M. Jabri. Handwritten digit recognition by adaptive-subspace self-organizing map (ASSOM). *IEEE Transactions on Neural Networks*, 10(4):939–945, 1999.

Exploring a Two-Market Genetic Algorithm

Steven O. Kimbrough*
Ming Lu
University of Pennsylvania
Operations & Information Man. Dept.
Philadelphia, PA 19104

David Harlan Wood
CIS Dept.
University of Delaware
Newark, DE 19716

D.J. Wu
LeBow College of Business
Drexel University
Philadelphia, PA 19104

Abstract

The ordinary genetic algorithm may be thought of as conducting a single market in which solutions compete for success, as measured by the fitness funtion. We introduce a two-market genetic algorithm, consisting of two phases, each of which is an ordinary single-market genetic algorithm. The two-market genetic algorithm has a natural interpretation as a method of solving constrained optimization problems. Phase 1 is optimality improvement; it works on the problem without regard to constraints. Phase 2 is feasibility improvement; it works on the existing population of solutions and drives it towards feasibility. We tested this concept on 14 standard knapsack test problems for genetic algorithms, with excellent results. The paper concludes with discussions of why the two-market genetic algorithm is successful and of how this work can be extended.

1 Motivation & Experimental Setup

Genetic algorithms (GAs), and more generally evolutionary computation, have much to recommend them as heuristics for unconstrained optimization. These problems are often otherwise intractable and experience has yielded broadly successful results. The case of *constrained* optimization is more problematic for evolutionary computation and GAs in particular. In spite of considerable attention paid to the matter (see [2, 5, 6] for excellent reviews), there is no clearly best approach to encoding constrained optimization problems as GAs. Techniques are available and used, largely based on penalty functions. Still, this important class of problems remains somewhat recalcitrant. We explore a conceptually new approach to constrained optimization, at least in the GA context.[1] We interpret a constrained optimization problem as a market between two players adapting as GAs: the objective function player and the constraint player. We explore how their markets behave (and how good the solutions are they find), compared to standard GA approaches for handling constraints. We use 14 well-studied knapsack test problems for our benchmarking [8]. Although the knapsack is perhaps the simplest of integer constrained optimization problems, it is NP-complete. Thus, we may hope its lessons apply to other problems of interest.

1.1 Penalty Functions & Fitness Evaluation

Given a solution, $\vec{s}$, to a constrained optimization problem, its absolute fitness, $W(\vec{s})$, in the presence of penalties for constraint violation is commonly ("standardly") measured as:

$$W(\vec{s}) = Z(\vec{s}) - P(\vec{s}) \tag{1}$$

where $Z(\vec{s})$ is the objective function value produced by $\vec{s}$,[2] and $P(\vec{s})$ is the total penalty (if any) associated with constraint violations by $\vec{s}$. We employed two widely-used penalty functions in our investigations. Both assume that constraint i has the form: $\sum_{j=1}^{n} a_{i,j}x_j \leq b_i$. (The x_js are the decision variables.)

sum-of-violations. With the α_is as weights $P(\vec{s}) = \sum_{i=1}^{m} \alpha_i \max\{0, \left(\sum_{j=1}^{n} a_{i,j}x_j - b_i\right)\}$ (We set $\alpha_i = 1$.)

violation-product. $P(\vec{s}) = s \cdot \max\{c_i\}$, where s is the number of violated constraints and $\max\{c_i\}$ is the largest objective function coefficient. This is the method used by Khuri [4] and others.

[1] We note a certain analogy between large-scale decomposition methods in optimization theory and our two-market GA.

[2] Or a linear transformation for computational convenience.

1.2 The Two-Market GA

In a GA the individuals in the population compete for success in a single market, driven by the fitness function and the consequences of the genetic operations. In our two-market GA (for constrained optimization problems), two single-market GAs alternate in acting upon a single (evolving) population of solutions. A single two-market generation consists of a phase 1 single-market treatment, "optimality improvement", followed by a phase 2 single-market treatment, "feasibility improvement".

During phase 1 of the two-market GA, a single-market (standard) GA is conducted using only the unconstrained objective function, Z, as the fitness function. Upon completion of phase 1, phase 2 commences. Phase 2 feasibility improvement uses a single-market GA whose fitness function is a penalized version of the problem's constraint set. All solutions meeting the constraint set are given an absolute fitness of 0. Any solution, $\vec{s}$, violating one or more constraints receives an absolute fitness of $-P(\vec{s})$, where P is the penalty function in use. In our experiments we investigated two such functions, as indicated above: (a) sum-of-violations (b) violation-product.[3]

In our experiments, one two-market generation consists of one optimality improvement generation followed by one feasibility improvement generation.[4] Specifically, our experiments ran for 500 generations. Our two-market GAs ran 250 phase 1 generations and 250 phase 2 generations. In contrast, we ran standard GAs (used for comparison) for the full 500 generations. Thus, the number of fitness evaluations were equal between our experimental two-market GAs and the standard GAs.

1.3 Benchmark Problems

We investigated 14 knapsack test problems from a standard GA source site [8]. The problems and their basic characteristics are indicated in Tables 1 and 2. We conducted experiments on the 14 knapsack test problems with 5 GAs: ksR, ks, ks2, ksc, and ksc2. ksR is a standard, one-market GA, using a repair approach to the 14 knapsack problems; it is present mainly for comparison purposes. The remaining four cover the four combinations of one- vs. two-market GA, and sum-of-violations vs. violation-product penalties, as follows:

Problem Name	Number of Variables	Number of Constraints	Optimal Value
hp2	35	4	3186
pb6	40	30	776
pb7	37	30	1035
pet7	50	5	16537
sento1	60	30	7772
sento2	60	30	8722
weing7	105	2	1095445
weing8	105	2	624319
weish12	50	5	6339
weish17	60	5	8633
weish21	70	5	9074
weish22	80	5	8947
weish25	80	5	9939
weish29	90	5	9410

Table 1: Test Problems

	sum-of-violations	violation-product
one-market	ks	ksc
two-market	ks2	ksc2

Thus, the ks-ks2 and ksc-ksc2 pairs are directly comparable because they use the same penalty functions.

We experimented with population sizes of 50, 500, and 5000. Results were comparable; we focus here on population size 5000. The number of generations was 250×2 for the two-market GAs, and 500×1 for the standard one-market GAs (which are used for comparison). Single-point crossover was used with a rate of 0.6, and the mutation rate was 0.1. (Sensitivity runs did not raise any anomalies.)

1.4 Initialization of Populations

We did use an innovative approach to initializing the populations. Commonly, populations are initialized with 1s and 0s by picking a probability that a site is a 1 and drawing random numbers to make the specific assignments. Khuri et al. [12] call this the *biased* approach to initialization. One has to wonder how the choice of that probability affects the outcomes, especially when different methods are being compared. We note that in the weing8 problem, Khuri et al. [12] have to use a biased random initialization of the population such that the probability of a zero at a site is 0.95. This allows, they discovered empirically, the initial population to have enough feasible solutions for

[3] Clearly, many variants, indeed convex combinations, of this two-market approach are possible and worth investigating. Each phase might take into account some constraints, the constraint set might be parcelled out to several phases, and so on.

[4] It would be interesting to investigate whether it pays to run more than one one-market generation per two-market generation.

their algorithm to work. The feasible region in this problem is extremely sparse. This biased approach introduces two concerns. First, the biased population may favor searching only initially and may lead to a sub-optimal solution. A second concern is that choosing the right probability is not trivial.

We introduce an alternate, non-empirical method of randomly initializing a GA's population. Khuri et al. claim the purpose of the biased initial population is to make sure the initial population has feasible solutions. Our alternative method for population initialization is very likely to have feasible solutions with large population size.[5] Our non-empirical procedure to generate the initial population is given in Figure 0.

```
Set incr, popsize; Prob=0; popsofar=0;
Do until popsofar==popsize {
   Produce one solution based on Prob;
   Put the solution into the population;
   popsofar++; Prob=Prob+incr;
   If Prob > 1, Prob=incr;
}//End of Do until we reach the population size
```

Figure 0: Procedure to generate an initial population. `Prob`, probability of a 1 bit in a solution; `incr`, small number to increment `Prob` for each solution; `popsize`, total size of population; `popsofar`, number of solutions generated. In our experiments, `incr` = (# of variables)$^{-1}$.

2 Results

Our results for populations of 5000 are summarized in Table 2. These data are qualitatively similar to those we got for populations of 50 and 500, except that there is a general trend, as expected, towards better solutions with larger populations.

We draw the reader's attention to Table 2 as follows. Column 1 reports the results of ksR, a single-market (standard) GA using repair to maintain a population of feasible solutions. ksR is useful for comparison purposes, but its computational cost was at least an order of magnitude greater than the other 4 methods.

We shall focus on columns 2-5. Columns 2 and 3 are directly comparable, as are columns 4 and 5. Columns 2 and 3 report means and standard deviations for runs using a penalty function based on the sum of the violations of the constraints. Column 2 is the standard, single-market GA; column 3 is the two-market GA. Columns 4 and 5 report means and standard deviations for runs using a penalty function based on the number of constraint violations times the maximum objective function coefficient. The means and standard deviations are for the best feasible solution in the 500th generation, across 5 runs using differently-seeded random number streams.

Examination of Table 2 reveals a general pattern: As we move from columns 2 to 3, and from 4 to 5, the means increase and the standard deviations decrease. That is, the two-market GAs typically find (on average, across 5 runs) a better feasible solution and do so with a lower variance. (A standard deviation of 0 indicates that in all 5 runs the best solution in the last generation was the same.) The comparison is summarized in column 0, with the +/-/M notation. Of the 14 problems, in just two cases—`pb6` and `pb7`—the two two-market GAs did (slightly) worse than the two one-market GAs. In one case—`weing7`—the results were mixed: on one regime of constraint violation the two-market GA did better and on the other it did worse, albeit just slightly so. (We also note that in `hp2`, ksc and ksc2 both find the optimal solution. Since ks2 does significantly better than ks, we award the two-market GA the +.) Finally, in 5 of 14 cases the two-market GA did substantially better (++) than the one-market.

Even though it is not possible to have a random sample of knapsack problems, statistical-style reasoning nonetheless offers some insight.[6] We can take the null hypothesis as stating that the two-market GA should get a + as often as the one-market GA. Charitably, we credit the M to the one-market GA, giving it a total of 3 victories in 14 trials. If the null hypothesis is true, the probability of getting 3 or fewer successes in 14 trials is $\sum_{x=0}^{3} \binom{14}{x} \left(\frac{1}{2}\right)^{14} = 0.029$ which is better than the generally accepted standard of 5%. That there should be 5 of 14 cases highly favorable to the two-market GA and 0 of 14 similarly favorable to the one-market GA is highly improbable, if the null hypothesis is true.

We note further that there are 3 smaller problems—`hp2`, `pb6` and `pb7`—having 35, 40 and 37 variables respectively. These are the ones on which the one-market GA did generally better, although the differences in its favor are not great. The other 11 problems had

[5] Our procedure generates an all-0 solution, which is certain to be feasible in the case of knapsack. The larger point is that if initializing a population with a constant value for `Prob` will yield feasible solutions in the neighborhood of some felicitous value, then our procedure will automatically produce some initial solutions with `Prob` set near that value. Further research is required to ascertain the value of our method; it certainly worked well in the present cases.

[6] For the record, we are not censoring any of our results. We examined all and only the 14 problems reported here.

between 50 and 105 variables. Among these problems, the two-market GAs did substantially better on 5, and clearly better on 5. On one, there was a tie.[7]

Thus, *for these knapsack problems* it appears that the two two-market GA is definitely superior to its two one-market analogs.[8] We do want to note, however, that four problems stand out in having larger numbers of constraints. On two of these—`pb6` and `pb7`—the standard one-market GA did better, and on two—`sento1` and `sento2`—the two-market GA did better. One explanation (cf. discussion below) is that what matters is how tightly the problem is constrained, not how many constraints it has. Perhaps it is easy to find feasible solutions for `pb6` and `pb7`, and harder for `sento1` and `sento2`. At this point, conjecture would be premature. We leave the issue for future research.

3 Discussion

What might explain why the two-market GA does so much better than the standard GAs, especially on the more complex problems (at least those with a higher number of variables)? A full answer requires both analytic and empirical work, which we are conducting. A conjecture, however, is worth discussing. The intuition may be explained as follows.

Recall from Expression (1) that given a solution, $\vec{s}$, to a constrained optimization problem, its absolute fitness, $W(\vec{s})$, in the presence of penalties for constraint violation is: $W(\vec{s}) = Z(\vec{s}) - P(\vec{s})$, where $Z(\vec{s})$ is the objective function value produced by $\vec{s}$, and $P(\vec{s})$ is the total penalty (if any) associated with constraint violation by $\vec{s}$.

We discuss three cases. They are distinguished by their positions with regard to what we shall call the (feasible) frontier. This is the border in fitness space between the feasible and infeasible regions.

[7]Even there, ks2 does better than ks (cf. the standard deviations), while the mean for ksc2 divided by the mean for ksc is: $\frac{1094409}{1094677} = 0.999755$. Perhaps we are too generous in crediting the one-market GA with a victory.

[8]We note that Khuri et al. [4], using a penalty regime comparable to ksc, report results for `sento1`, `sento2` and `weing7` that are very close to those we report in Table 2. Their result for `weing8` is 613383. They were forced to resort to a biased initialization to achieve this, however. "In the case of the [`weing8`] problem a biased random initialization of the population ... is used such that the probability to generate a zero bit is 0.95. This simple but elegant solution makes the problem more amenable to our genetic algorithm approach." And even simpler and more elegant would be to consult an Oracle for the right answer.

3.1 Case 1: Feasible and Far from Frontier

First, all members of our population are feasible and far from the (feasible) frontier. Here, all penalties will be zero and a GA will favor—in the special case of the knapsack—adding items to the knapsack having high objective function coefficients, regardless of how much of the constrained resources they consume. (The point is generally valid, beyond just the knapsack problem.) In this situation, the penalty functions do not come into play; the standard GA (with penalties) becomes simply a GA; and the two-market GA becomes a slower version of the standard GA. (On the last point, recall that for comparison purposes we do $\frac{1}{2}$ generation of two-market GA per generation of standard GA.) Thus, when all (or most) solutions are feasible and far from incurring penalties, we would expect the standard GA to outperform the two-market GA.

3.2 Case 2: Population Infeasible

The second case to consider is when all (or most) members of our population are infeasible. Let us assume (without, we think, any real loss of generality) that fitness proportional selection is being used and is roughly correct; i.e., assume we agree that fitness proportional selection is approximately correct in setting the incentives to the GA for its search. Keeping the example simple (but again, without essential loss of generality), consider a population with just two solutions, $\vec{s_1}$ and $\vec{s_2}$. Let's say that $\vec{s_1}$ is twice as good as $\vec{s_2}$, i.e., $Z(\vec{s_1}) = 2Z(\vec{s_2})$. When the penalties are zero (case 1), the relative fitnesses, are:

$$F(\vec{s_1}) = \frac{Z(\vec{s_1})}{(Z(\vec{s_1}) + Z(\vec{s_2}))} = \frac{2}{3} \qquad (2)$$

and $F(\vec{s_2}) = 1 - F(\vec{s_1}) = \frac{1}{3}$. What if both solutions are infeasible? Let us assume both solutions are equally infeasible, so that their penalties are identical (again, with essential loss of generality). Note that penalties, however they are set, need to be relatively large in order to drive the search towards feasible solutions. In general, if $P(\vec{s}) > 0$ (that is, if there is any constraint violation at all by $\vec{s}$) then $P(\vec{s}) \gg Z(\vec{s})$. Thus, with penalties kicking in, $F(\vec{s_1}) =$

$$\frac{(Z(\vec{s_1}) - P(\vec{s_1}))}{((Z(\vec{s_1}) - P(\vec{s_1})) + (Z(\vec{s_2}) - P(\vec{s_2})))} \approx \frac{1}{2}. \qquad (3)$$

The large P values overwhelm the Z values, greatly reducing the relative fitness differences between infeasible solutions. Penalty functions may be excellent at distinguishing feasible from infeasible solutions, but at the price of obfuscating the differences between infeasible solutions. Note a numerical example. Let:

0	1	2	3	4	5
Problem	ksR	ks	ks2	ksc	ksc2
hp2	3186	2948.6	3114.8	3186	3186
+	(0)	(16.772)	(14.307)	(0)	(0)
pb6	776	740.8	646.2	776	730.2
-	(0)	(19.176)	(45.252)	(0)	(17.283)
pb7	1034.8	994.2	966	1034.4	1033
-	(0.447)	(11.946)	(23.292)	(0.894)	(4.472)
pet7	16483.2	15811.8	16452	16457	16486.6
+	(20.969)	(169.428)	(11.726)	(38.085)	(21.984)
sento1	7743.2	7732.2	7758.2	7739.2	7769.8
+	(27.905)	(23.626)	(11.987)	(27.563)	(4.919)
sento2	8672.4	8669.2	8720.4	8671.4	8703.2
+	(20.379)	(19.690)	(3.050)	(16.410)	(3.701)
weing7	1089914	1084623	1094727	1094677	1094409
M	(268.319)	(3168.344)	(398.92)	(385.755)	(407.547)
weing8	619925	342959	611820.2	321133.8	623627.8
++	(1461.077)	(56711.01)	(6930.155)	(44832.01)	(867.579)
weish12	6339	6009.2	6338.8	5689.4	6339
+	(0)	(435.119)	(0.447)	(242.683)	(0)
weish17	8624.2	8630.6	8633	7692.6	8633
+	(4.919)	(5.366)	(0)	(522.115)	(0)
weish21	9053	8538	9013.4	5369.4	9074
++	(7.969)	(424.229)	(21.686)	(354.420)	(0)
weish22	8921.4	5575	8891.4	5451.2	8939.8
++	(22.03)	(436.584)	(20.403)	(189.254)	(9.859)
weish25	9904	9758.4	9903.6	6083	9939
++	(16.325)	(214.057)	(30.411)	(291.32)	(0)
weish29	9383.2	5425	9203.2	5068	7530.2
++	(9.859)	(215.431)	(116.154)	(267.685)	(315.906)

Table 2: ksR = one market GA (repair). ks = standard GA, penalty based on sum of violations. ks2 = 2 market GA, penalty based on sum of violations. ksc= standard GA, penalty based on number of violations × max coefficient. ksc2= 2 market GA, penalty based on number of violations × max coefficient.
The problem names are key to their original sources: `hp2`, `pb6`, and `pb7` [3]; `pet7` [7]; `sento*` [9]; `weing*` [12]; and `weish*` [10]. (In virtue of being knapsack problems, all 14 are maximization problems.)
Key: mean , (standard deviation); for best solution after 500 generations, population size 5000, five runs.
+ two-market GAs did better. ++ two-market GAs did much better. - two-market GAs did worse. M mixed.

$Z(\vec{s_1}) = 1$, $Z(\vec{s_2}) = 0.5$, $P = 2.25$. Then the relative fitnesses are $F(\vec{s_1}) = 1 - 0.417 = 0.583$ and $F(\vec{s_2}) = 1 - 0.583 = 0.417$ (switching signs for comparison purposes) compared to 0.667 and 0.333 without penalties. Even a small P value noticeably blurs the differences.

The two-market GA in its phase 1 (optimality improvement) sees the Z values of the solutions, unencumbered by the fog of penalties. Phase 2 of the two-market GA (feasibility improvement) edits the population in favor of feasibility. The process drives towards feasible optimality. The advantage of the two-market GA lies in its ability to respond more usefully to infeasible solutions. If you are going to have a population with a large number of infeasible solutions, the two-market GA is the GA for you.[9]

3.3 Case 3: Population on the Frontier

Finally, consider a third case: the solutions in the population are all (precisely) on the frontier (and thus feasible). In the case of the knapsack this means that setting any 0 to 1 (indeed increasing the value of any variable) will turn a feasible solution infeasible.

The standard GA sees the relative fitnesses clearly as in equation (2). The more optimal solutions (with higher Z values) will contribute proportionately more to the next generation. Since the GA operators (e.g., mutation and crossing over) are blind, a large percentage of the products of these operations will be infeasible, and receive very low fitness in the next generation. If the operations find a better schema (setting values higher for certain variables), the solution will be infeasible unless the values of other variables are reduced sufficiently to ensure feasibility. Often it will be the case that this does not happen; and the better schema has no real chance to be tried in the population.

Suppose instead that we are in case three, with solutions all on the frontier, and we are also in phase 2 of the two-market GA. Because the population is entirely feasible, the GA here sees each solution as equally fit. A new population is accordingly formed after application of the genetic operators. Again, suppose that the genetic operations find a better schema. Phase 1 will recognize it and with fitness proportional selection try it out in relatively more solutions in the new population it sends to phase 2. This mechanism (probabilistically) gives the better schemas more chances to locate themselves in feasible solutions. If it does, phase 2 will let it pass and the schema stands a chance of surviving.

The considerations of this section lead us to conjecture that the two-market GA will be superior to the standard, penalty-function GAs (on constrained optimization problems), when feasible solutions are hard to find and the GA must process many infeasible solutions. (Compare cases 1 and 2 above.) Further, we conjecture (case 3) that the two-market GA has a superior "end-game" performance. Given a population on the (feasible) frontier, the two-market GA provides a better opportunity for trying out advantageous schemas.

Table 2 provides some evidence in favor of the case 3 factor. Note that ksR, a one-market GA using a repair approach, maintains (at great computational expense) a feasible population. The GA will drive this population to the frontier. Because ksR keeps generating solutions to maintain a feasible population, it does not have the case 3 problem that the standard (penalty function) GA has in not being able to explore with new schemas once it gets to the frontier. For case 3, neglecting computational costs, we expect ksR and the two-market GA to perform similarly. We note in this regard that in, *and only in,* the five cases in which the two-market GA performs substantially better than the standard GA (++ in table 2), ksR also performs substantially better than the standard GA.

Suggestive evidence is also available in Figures 1 through 6, comparing the progress over generations of the 5 knapsack algorithms (ksR, ks, ks2, ksc, and ks2) as they work on `weish17` and `weing8`. The Figures show *typical* runs. `weish17` is a problem on which the two-market GA does slightly better. We see in the Figures ksc quickly reaches a plateau of 80% of optimality and gets stuck. ks does better and quickly reaches a plateau close to optimality, but not as quickly as ks2. `weing8` is a problem on which the two-market GA does substantially better than the standard GA. We see in the Figures that the two-market GAs level off near optimality by about 150 generations, while ksc and ks level off immediately at less than 60% of optimality and never improve. This *suggests* a case 3 situation.

On extensions to these ideas, we are particularly intrigued with the prospect of applying these results to non-standard computational regimes, especially DNA computing. Genetic Algorithms seem particularly suited to implementation in DNA Computing [11, 1]. An originating impetus for this work was our formulation of a bargaining problem as a two-market GA for DNA computing. In DNA computing, it is awkward

[9] We suspect that the problem of finding optimal penalty functions for incenting a standard GA cannot be solved a priori, that the information is not available prior to undertaking exploratory computations. Further, the two-market GA can be thought of as circumventing this problem by loosening the link between Z and P. These ideas require further space than is available here for their exposition.

to make computations that combine both the objective functions and the penalty functions—thus the incentive for a two-market approach. Happily, this paper reflects advantages of the two-market approach beyond its convenience for DNA computing. In turn, DNA is quite suitable for two-market GAs; the results here are encouraging.

DNA computing features massively parallel processing of huge populations of candidate solutions. Thus, our work has an interesting sidelight in that there was a clear benefit of increasing population size. We performed runs with population sizes of 50, 500, and 5000. The results for 50 and 500 were qualitatively like those in Table 2, except that the general quality of the solutions increased with population size. For example, for ksc2 and averaging across all 14 problems, the mean solution was 1.1% higher at 500 generations than it was at 50 generations, and 3.2% higher at 5000 generations than it was at 50 generations. For the two-market GAs, excellent results were obtained in much less than 500 generations.

This has been an exploratory study. Much remains to be done by way of testing our conjectures and investigating the two-market GA, including: study of more knapsack problems, extension to other kinds of constrained optimization problems, deepening and refining mathematically the intuitions behind the three cases, extending the 2-market case to N-markets, and looking carefully at the GA histories. Finally, we remark that all this is most encouraging for DNA computing. Indeed, the results suggest many avenues of fruitful exploration.

Acknowledgments

This material is based upon work supported by, or in part by, DARPA contract DASW01 97 K 0007. GA202.

References

[1] Junghuei Chen and David Harlan Wood. Computation with biomolecules. *Proceedings of the National Academy of Sciences, USA*, 97(4):1328–1330, 2000. Commentary.

[2] Carlos Artemio Coello Coello. A survey of constraint handling techniques used with evolutionary algorithms. technical report Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada, Veracruz, México, 1999. http://www.lania.mx/~ccoello/constraint.html.

[3] A. Freville and G. Plateau. Hard 0-1 multiknapsack test problems for size reduction methods. *Investigation Operativa*, 1:251–270, 1990.

[4] Sami Khuri, Thomas Bäck, and Jörg Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In *Proc. of the ACM Symp. of Applied Comp. (SAC'94)*. ACM Press, 1993.

[5] Zbigniew Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155, Cambridge, MA, 1995. MIT Press. http://www.coe.uncc.edu/~zbyszek/papers.html.

[6] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, Germany, third edition, 1996.

[7] C. C. Petersen. Computational experience with variants of the Balas algorithm applied to the selection of R&D projects. *Management Science*, 13:736–750, 1967.

[8] CMU Artificial Intelligence Repository. SAC94 Suite: Collection of multiple knapsack problems. World Wide Web, Accessed January 2002. http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/test/sac/.

[9] S. Senyu and Y. Toyada. An approach to linear programming with 0-1 variables. *Management Science*, 15:B196–B207, 1967.

[10] W. Shi. A branch and bound method for the multiconstraint zero one knapsack problem. *J. Opl. Res. Soc.*, 30:369–378, 1979.

[11] Willem P. C. Stemmer. The evolution of molecular computation. *Science*, 270:1510–1510, December 1, 1995.

[12] H. M. Weingartner and D. N. Ness. Methods for the solution of the multi-dimensional 0/1 knapsack problem. *Operations Research*, 15:83–103, 1967.

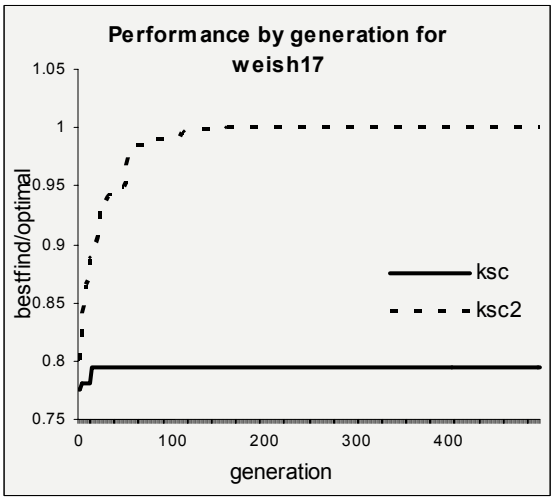


Figure 1: ksc vs. ksc2 for weish17

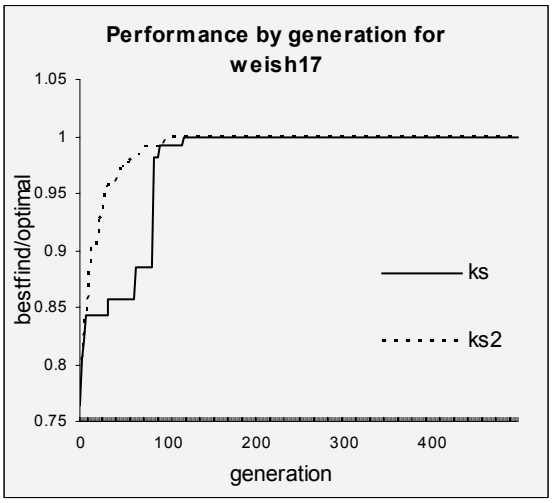


Figure 2: ks vs. ks2 for weish17

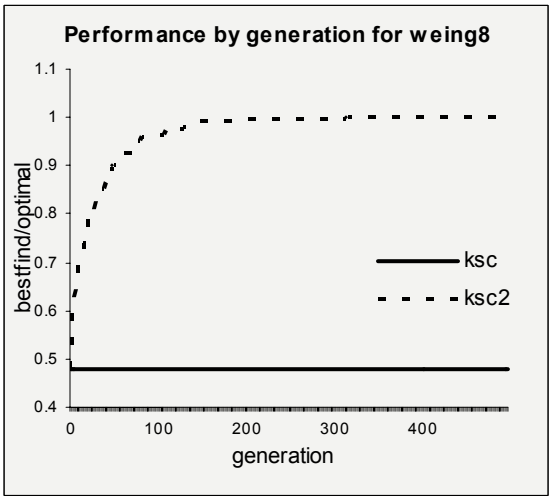


Figure 3: ksc vs. ksc2 for weing8

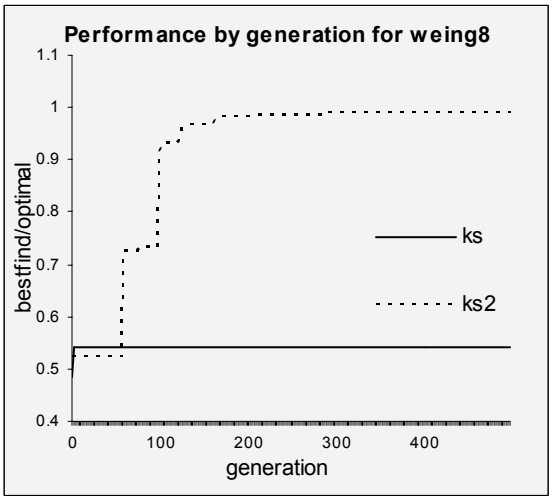


Figure 4: ks vs. ks2 for weing8

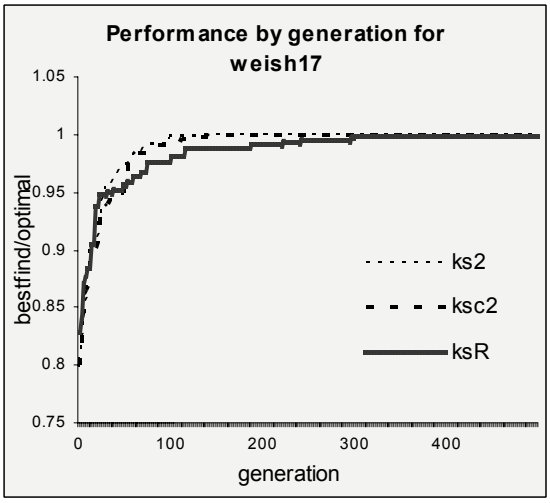


Figure 5: ks2 and ksc2 vs. ksR for weish17

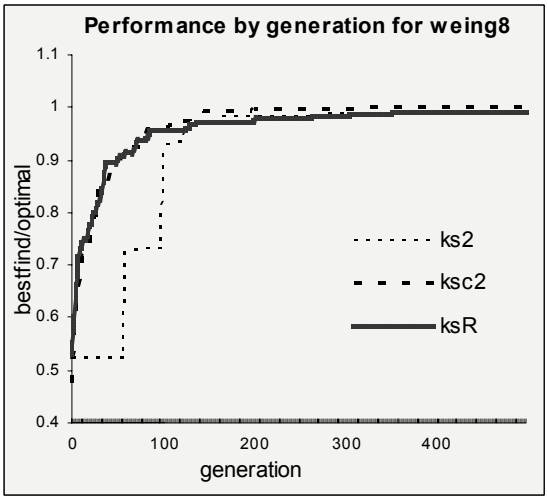


Figure 6: ks2 and ksc2 vs. ksR for weing8

MOCS: Multi-Objective Clustering Selection Evolutionary Algorithm

Thomas E. Koch and Andreas Zell
University of Tübingen, WSI/RA
Sand 1, D-72076 Tübingen, Germany
{koch,zell}@informatik.uni-tuebingen.de

Abstract

In this paper, we describe a multi-objective evolutionary algorithm, that uses clustering selection and does not need any additional parameter like others. It clusters the population into a flexible number of clusters employing x-means from [Pelleg and Moore, 2000]. First, the selective fitness is assigned to clusters and in second place to individuals of clusters. We show three hybrid variants incorporating additional mechanisms from other elitist multi-objective evolutionary algorithms in order to increase selection pressure. Using the test functions from Deb's T suite (T1-T6), from Schaffer, Kursawe and Quagliarella we evaluate the performance and the quality of our approach against the most recent and performant elitist multi-objective evolutionary algorithms, NSGA2, SPEA2 and PESA2. The comparison yields promising results for region-based selection using clustering in combination with additional crowding strategies.

1 Introduction

Multi-Objective Optimisation Problems (MOOP) arise when at least two competing objectives (or criteria) have to be optimised. If no preferences for any objectives are given or known a priori, the task is to optimise all objectives at the same time, which will produce a set of optimal trade-off solutions rather than one single optimal solution. In general a MOOP is defined by a function f

$(y_1, y_2, ..., y_m) = f(x_i, x_2, ..., x_n)$ with $m > 1, n > 0$

which maps a vector of n decision variables to a vector of m objective variables, which has to be optimised. Solutions may be better, equal or worse than others. Better or *dominating* solutions are meant to be better in at least one objective and not worse in all others. All non-dominated solutions are called *Pareto-optimal* and belong to the *Pareto set*.

Most Evolutionary Algorithms (EA) try to solve MOOPs by using a priori knowledge —given or not— to weight the different objectives in order to construct a single objective problem. But these approaches produce just one solution of the Pareto-optimal set. In contrast, Evolutionary Multi-Objective Optimisation Algorithms (EMO) try to find the whole Pareto set or at least a good presentation of it. For further reading in MOOP and EMO we suggest the comprehensive book from Deb [Deb, 2001].

Since the first real EMO algorithm by Schaffer [Schaffer, 1984], called VEGA, and the inspiring lines in Goldberg's book [Goldberg, 1989], a number of seminal approaches have shown the capability of EMOs to demonstrate that Pareto domination-based EMOs can be reliably used to find and maintain multiple trade-off solutions of the Pareto set. In the last years elitist EMOs have shown best performance in order to find global Pareto-optimal solutions and good diversity in presenting the real global Pareto set: *Non-dominated Sorting Genetic Algorithm 2* (NSGA2) [Deb et al., 2000], *Strength Pareto Evolutionary Algorithm 2* (SPEA2) [Zitzler et al., 2001] and *Pareto Envelope based Selection* (PESA) [Corne et al., 2000] and PESA2 [Corne et al., 2001]. All these algorithms use the framework of conventional EAs and differ in fitness assignment, selection operator and an optional external archive, storing the actual Pareto set.

In the following of the paper we show a new Multi-Objective Clustering Selection operator (section 2), compare it to the mentioned algorithms on commonly used test problems (section 3), discuss the results (sec-

tion 4), and conclude (section 5).

2 Multi-Objective Clustering Selection (MOCS)

The Multi-Objective Clustering Selection (MOCS) Evolutionary Algorithm works as follows:

Algorithm 1 (MOCS EA) Population P_t of $\lambda \leq N$ individuals is evolved for T generations. An additional archive population P_t^* of size $\mu = N$ is maintained. P_T^* gives the result of non-dominated solutions.

1. *Initialisation*: Set $t = 0$, generate initial offspring population P_t, set population $P_t^* = \emptyset$.
2. *Evaluate and Assign Fitness*: Evaluate fitness values of individuals of P_t.
3. *Environmental Selection*: Use a truncation operator in order to reduce the size of $P_t^* = P_t^* \cup P_t$ to N (cf. section 2.1): We use "non-dominated Pareto sorting" and "crowding distance measure" from NSGA2.
4. *Termination*: If $t \geq T$ then remove all dominated individuals from P_T and stop.
5. *Mating Selection*: **Cluster** P_t^* into k clusters using $(c_1, c_2, ..., c_k) = x\text{-}means(P_t^*, 1, N)$ with $1 \leq k \leq N$ [Pelleg and Moore, 2000] (cf. section 2.2).
   (a) Region-based selection: Perform binary tournament selection on the k clusters found where clusters with lower cardinality win (like PESA does) $\Rightarrow \lambda$ selected clusters.
   (b) Local individual-based selection: For any of the λ selected clusters perform a binary tournament selection on its individuals where the crowding-distance measure from NSGA2 orders individuals $\Rightarrow \lambda$ selected individuals.

   The mating pool P_t contains now λ individuals.
6. *Variation*: Apply recombination and mutation to P_t.
7. *Increment*: Set $P_{t+1}^* = P_t^*$ and $P_{t+1} = P_t$. Set $t = t + 1$ and go to 2.

As alternatives step 5a is replaced with random selection of clusters and step 5b is replaced with random selection of individuals (PESA-like). Table 1 gives an overview of our implemented combinations.

MOCS uses the basic Genetic Algorithm extended by *Environmental Selection* and *Mating Selection* like other elitist EMOs do. The archive is limited by N entries and has to be reduced in the *Environmental Selection* step, because the offspring population has to be merged into the archive, which yields a maximum size of $2N$. It is important to keep as many non-dominated solutions as possible and a good diversity among them in the archive. In the *Mating Selection* step individuals are chosen for the mating pool. Again the best and most diverse solutions should be selected with higher probability in order to increase selection pressure for finding better solutions in the variation step.

Besides its own clustering technique, MOCS uses ordering techniques from other EMOs (cf. table 1): PESA2 [Corne et al., 2001] has a "squeeze"-factor, which simply counts the individuals of each hyperbox (see also section 2.2). We use this in a similar way: In a binary tournament the cluster with less members wins. From NSGA2 [Deb et al., 2000] we borrow the "non-dominated Pareto sorting" and the "crowding distance measure" which orders individuals firstly by its Pareto-rank (lower is better) and secondly by the volume enclosed by its next neighbours (larger is better). In NSGA2 it is used for environmental and mating selection.

Algorithm	Mating Selection (after clustering with X-means)
MOCS-1	randomly select clusters, then perform binary tournament with crowding-distance measure
MOCS-2	perform binary tournament on clusters with "squeeze"-factor, then perform binary tournament with crowding-distance measure
MOCS-3	perform binary tournament on clusters with "squeeze"-factor, then randomly select individuals from winning clusters

Table 1: Overview of implemented versions: MOCS uses the crowding-distance measure from NSGA2 and the "squeeze"-factor from PESA2 in order to rank individuals resp. clusters.

2.1 Environmental Selection

In a first step we remove all multiple points and keep just one of each. This may reduce the population (archive) size below N but gives all points the same chance in the later steps of the algorithm. MOCS uses Environmental Selection from NSGA2 in order to reduce the population (archive) size. We also implemented and tested a clustering technique for En-

vironmental Selection similar to the NSGA2, but we did not include it in this paper: First do a non-dominated Pareto ranking, which assigns all individuals to r Pareto fronts; then successively add the ranked fronts of individuals until the actual front does not fit into the remaining s of N slots. For this actual front first keep all boundary points b if possible then perform a clustering into $s - b$ clusters and take the best individual from each cluster into the population. Best individuals are those who have the maximum average distance to all others. Remaining fronts are discarded.

2.2 Mating Selection

Mating selection in MOCS is done region-based. Corne and Knowles introduced this technique in EMOs with PESA2 [Corne et al., 2001]. They showed that region-based selection has advantages over individual-based selection like all other elitist EMOs do: the probability of selecting highly isolated individuals in contrast to crowded individuals rises because the unit of selection is now the "region" and no longer the individual. Thus this technique removes selective attention from crowded regions and assigns this attention more equally over the whole population.

The drawback on PESA2 is its 'hypergrid'. The user has to provide a parameter called grid-size g in order to build g^m hyperboxes in the m-dimensional objective hyperspace. The difficulty is to choose the best dimensions of the hyperboxes so that the resulting hyperboxes or regions are neither too fine-grained nor too large. The first extreme leads back to individual-based selection and the second extreme could lead to one hyperbox containing all individuals. Big advantages of PESA2 and the hypergrid strategy are its low performance complexity of $O(mN)$ to find the hyperbox for every individual per generation and the easy implementation of the algorithm.

MOCS uses a more flexible but also more expensive technique: we cluster the population into k clusters, where k is determined by the algorithm itself. k may be in the range from 1 to population size N. The used clustering algorithm *x-means* was introduced by [Pelleg and Moore, 2000]. X-means extends and improves *k-means* [Duda and Hart, 1973], which clusters a dataset in k clusters. For detailed explanations of x-means and k-means we refer to the cited papers, figure 1 shows an example.

Related work has been done by [Molyneaux et al., 2001]: They introduced the Clustering Pareto Evolutionary Algorithm (CPEA), which finds and retains many local Pareto-optimal fronts in contrast to our global Pareto-optimal algorithm. Major drawbacks of CPEA are the unlimited number of non-dominated (local) individuals, the fixed number k of clusters, the local non-dominated sorting (which obviously may be an advantage in the sense of searching for local Pareto-optimal sets) and its poor computational performance due to the unlimited size of the archive.

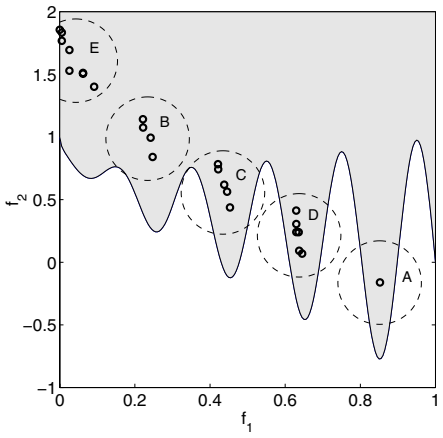


Figure 1: Population of 25 individuals in an EA run for test function T3 (cf. table 2, minimise both, f_1 and f_2): 5 clusters (A-E) have been calculated. In the Mating Selection of this population the cluster A down right has the best chances in the tournament selection, because it has the lowest number of elements.

2.3 Complexity Issues

The most complex task in multi-objective optimisation is to find an adequate process to calculate the selective fitness in the sense of crowding in objective space. Isolated individuals must have a higher selective fitness than crowded ones. This selective fitness is used for environment and mating selection to increase selective pressure.

PESA2 has a complexity of $O(mN)$ to calculate the box of N individuals and the "squeezes" of the boxes in an m-dimensional problem per generation. NSGA2 and SPEA2 require $O(mN^2)$ time.

Simple k-means has a performance complexity of $O(mNk)$ to calculate all distances between the N individuals and the k chosen centroids per internal iteration, where the number of internal iterations is not known and may be infinite. X-means is approximately equivalent to running k-means with $k = 1, 2, 3...N$ and uses kd-trees to store data which are much more efficient than the naive algorithm (for our low dimensionality). But anyway this yields a complexity of $O(mN^2)$

Name	Domain	Chromosome Length L	Functions
[Zitzler et al., 2000] [Deb, 2001]			
T1-T6 common frame			$f_1(x)$ $f_2(x) = g(x)h(f_1(x), g(x))$
T1	$[0,1]^n$ $n = 30$	900	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$ $h(f_1, g) = 1 - \sqrt{f_1/g}$
T2	$[0,1]^n$ $n = 30$	900	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$ $h(f_1, g) = 1 - (f_1/g)^2$
T3	$[0,1]^n$ $n = 30$	900	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$ $h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1)$
T4	$x_1 \in [0,1]$ $x_{i\neq 1} \in [-5,5]$ $n = 30$	300	$f_1(x) = x_1$ $g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}(x_i^2 - 10\cos(4\pi x_i))$ $h(f_1, g) = 1 - \sqrt{f_1/g}$
T5	$x_1 \in \{0,1\}^{30}$ $x_{i\neq 1} \in \{0,1\}^5$ $n = 11$	80	$u(x_i)$ denotes the number of 1s in x_i $f_1(x) = 1 + u(x_1)$, $g(x) = \sum_{i=2}^{n} v(u(x_i))$, $h(f_1, g) = 1/f_1$ $v(u(x_i)) = 2 + u(x_i)$ if $u(x_i) < 5$ else 1
T6	$[0,1]^n$ $n = 10$	300	$f_1(x) = 1 - \exp(-4x_1)\sin^6(6\pi x_1)$ $g(x) = 1 + (n-1)\left[\left(\sum_{i=2}^{n} x_i\right)/(n-1)\right]^{0.25}$ $h(f_1, g) = 1 - (f_1/g)^2$
[Schaffer, 1985]			
SPH-m $m = 2, 3, 4$	$[-10^3, 10^3]^n$ $n = 20$	400	$f_j(x) = (x_j - 1)^2 + \sum_{1\leq i\leq n, i\neq j} x_i^2$ $1 \leq j \leq m$
[Quagliarella and Vicini, 1997]			
QV	$[-5,5]^n$ $n = 20$	400	$f_1(x) = \left(\frac{1}{n}\sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i) + 10)\right)^{0.25}$ $f_2(x) = \left(\frac{1}{n}\sum_{i=1}^{n}((x_i - 1.5)^2 - 10\cos(2\pi(x_i - 1.5)) + 10)\right)^{0.25}$
[Kursawe, 1991]			
KUR	$[-10^3, 10^3]^n$ $n = 20$	400	$f_1(x) = \sum_{i=1}^{n-1}(-10\exp(-0.2\sqrt{x_i^2 + x_{i+1}^2})$ $f_2(x) = \sum_{i=1}^{n}(\lvert x_i\rvert^{0.8} + 5\sin^3(x_i))$

Table 2: Test functions for performance comparisons. Deb's T test suite, QV and KUR are 2-dimensional, SPH-m is evaluated in 2, 3 and 4 dimensions.

in best case.

3 Experiments

The MOCS approaches have been tested against NSGA2, SPEA2 and PESA2[1]. Table 2 shows the test functions with domains of decision variables, dimension of variable space n, dimension of objective space m and chromosome length L. All of them are minimisation problems. We used the same functions T1-T6 like [Corne et al., 2001] did for testing PESA and PESA2, like [Zitzler et al., 1999] did for testing SPEA and 8 other EMOs and KUR, QV and SPH-m like [Zitzler et al., 2001] did for testing SPEA2. But also many other researchers have used these functions as well.

Deb [Zitzler et al., 2000, Deb, 2001] provides a procedure of constructing two-dimensional objective problems with a range of characteristics of varying degrees. These include convexity (T1), concavity (T2), discontinuity (T3,T5), multi-modality (T4), deception (T5) and non-uniformity (T6) at the Pareto front. All these problems have to be managed in multi-objective real world problems by optimisers. T1 and T2 are the baseline tests. T1 is convex and thus a simple hill-climber would do the best job. T2 is concave, which yields first difficulties to overcome. T3 has a number (in

[1] We also tested against PESA and SPEA, but their performance was too poor, which has also been proven by the evaluation of its successors.

this case 5) of disconnected Pareto-optimal fronts. T4 has a convex Pareto-optimal global front but furthermore there exist $8 \cdot (10^{11})$ local fronts which produce a large number of hurdles. T5 is a boolean function over bit strings, has again many local fronts and attempts to deceive in order to lead the algorithm to a local instead to the global front. T6 is non-convex, non-uniform and the density to the global front is thin. We ran 2000 generations, which yields a total of 200K fitness evaluations in place of 5K fitness evaluations in other studies. We think looking at the whole optimisation process gives more insight into the behaviour of the algorithms. We tested all the algorithms with the same parameter settings [Corne et al., 2001](PESA2) and [Zitzler et al., 1999](SPEA) did for testing their algorithms.

SPH-m is a multi-objective generalisation of the sphere model [Schaffer, 1985], which is a symmetric unimodal function where the isosurfaces are given by hyperspheres. We used versions with two, three and four objectives. QV [Quagliarella and Vicini, 1997] consists of two multi-modal functions, an extreme concave Pareto-optimal front and a diminishing density of solutions towards the Pareto front. KUR [Kursawe, 1991] consists of a multi-modal function and function with pair-wise interactions among the variables, the Pareto front is disconnected consisting of concave and convex parts and an isolated point. Again we used the same parameter settings [Zitzler et al., 2001] did for testing SPEA2.

All algorithms are implemented in Matlab, embedded in a binary-coded Genetic Algorithm framework, but Evolution Strategies or real-coded Genetic Algorithms may be used as well. Table 3 shows the parameter setting for the Genetic Algorithm. For each algorithm and each problem, 20 runs with different random seeds have been evaluated and per run 200 intermediate results over time have been measured.

For measuring the quality of the results we have employed the hypervolume approach. The hypervolume approach by [Zitzler et al., 1999] (modified in [Zitzler et al., 2001]) calculates the portion of the normalised non-dominated hyperspace in a constructed hyperspace. Zitzler e.a. state it as the most appropriate scalar indicator since it combines both the distance of solutions (towards some utopian trade-off surface) and the spread of solutions.

Population sizes ($\mu = archive$)	T1-T6: $\mu = 100, \lambda = 10$ KUR, QV, SPH-m: $\mu = \lambda = 100$
#Generations	T1-T3,T6: 20000, T4,T5: 4000 KUR, QV, SPH-m: 10000
Crossover method Crossover rate p_c	uniform 0.7
Mutation rate p_m	$1/L$, where L (bit-wise)
Additional parameters	32x32 hypergrid used in PESA, PESA2

Table 3: Parameter settings for Genetic Algorithm

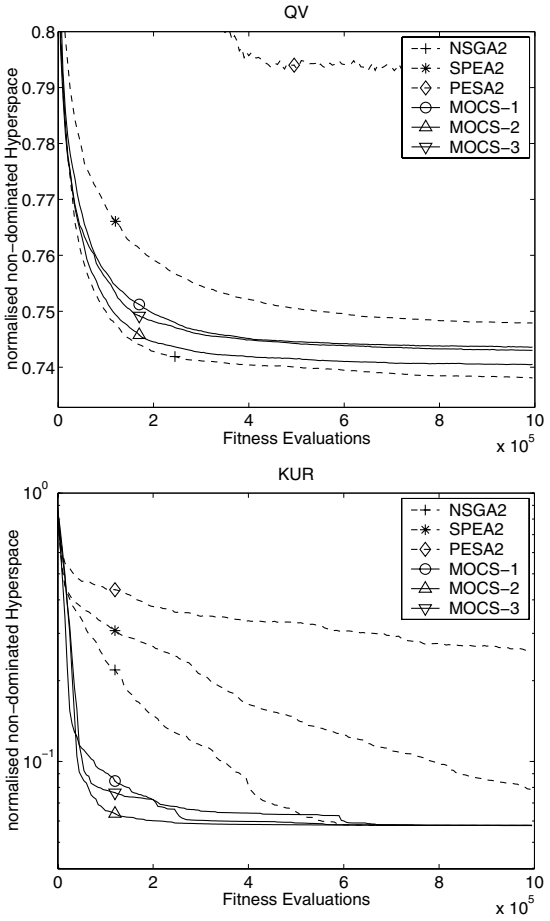


Figure 2: Performance values for QV and KUR. The graphs show the average normalised non-dominated portion of the hyperspace for 20 runs.

4 Results and Discussion

Figures 3 (T1-T6), 2 (KUR,QV) and 4 (SPH-m) show the results of all runs over time. All algorithms reach the Pareto fronts for the T functions except for T5, where they are deceived and get stuck. On T1-T6 all algorithms except PESA and PESA2 perform quite well and show similar curves. PESA and PESA2 lack the ability of keeping boundary points. They also do badly after reaching the global Pareto-optimal front, where it is crucial to spread uniformly over the front

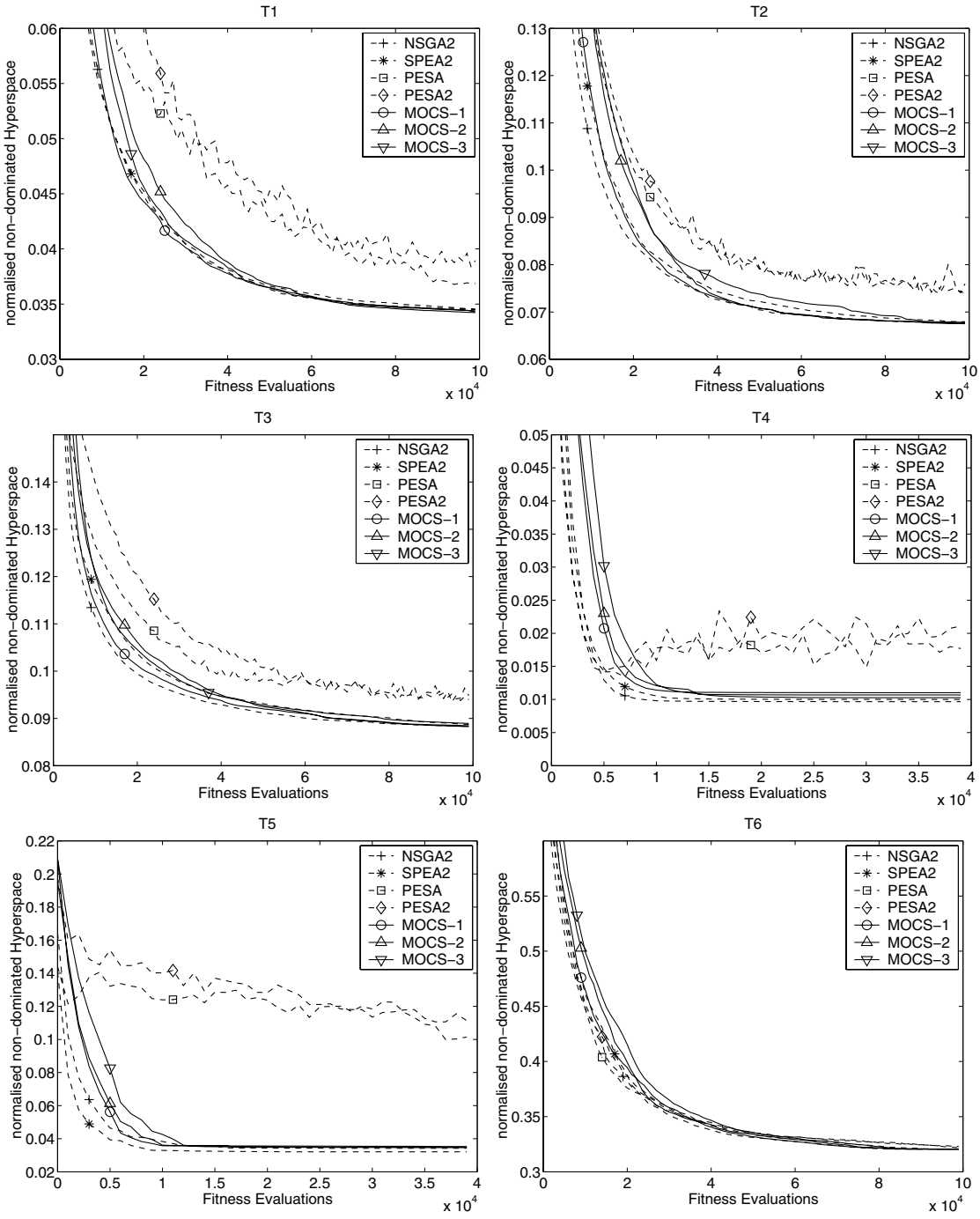


Figure 3: Performance values for T1-T6. The graphs show the average normalised non-dominated portion of the hyperspace for 20 runs.

in order to lessen the non-dominated hyperspace. The results for the T suite also shows that NSGA2, SPEA2 and our approaches are better than PESA and PESA2. On the T suite test functions our approaches keep track with NSGA2 and SPEA2.

On the QV, KUR and SPH-2 functions also all Pareto fronts are reached by MOCS. Again PESA2 performs worst and stagnates because boundary points are not kept. On QV NSGA2 is slightly better than our approaches. On KUR our approaches all perform quite

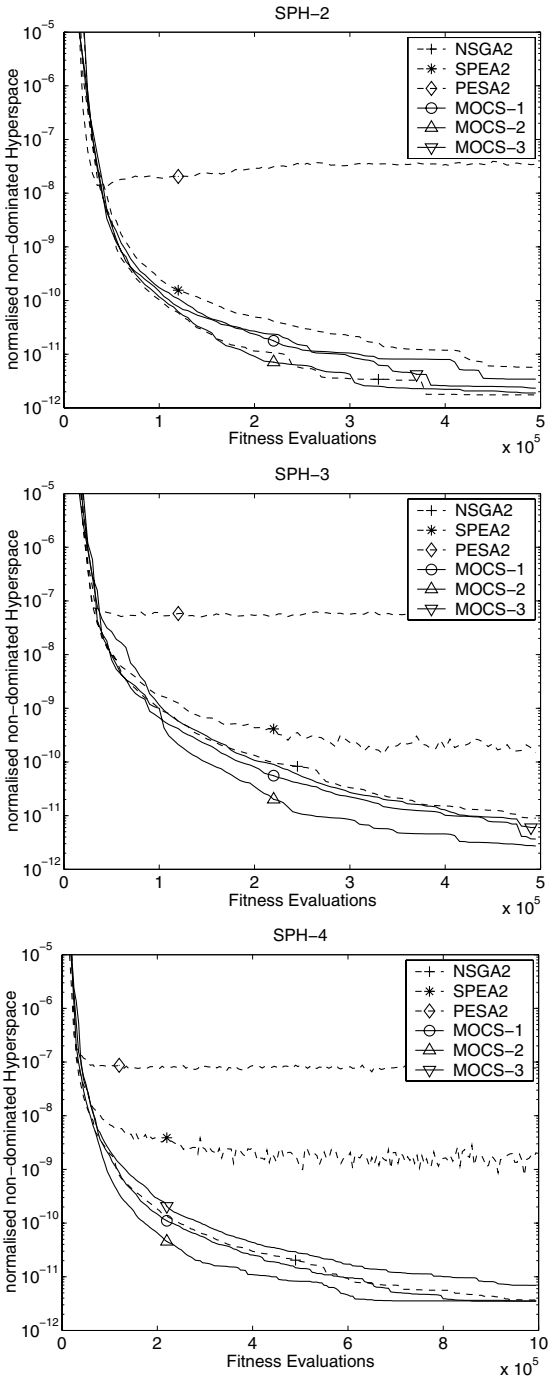


Figure 4: Performance values for SPH-[2,3,4]. The graphs show the average normalised non-dominated portion of the hyperspace for 20 runs.

better than all others. This is due to the fact that our first fronts are always ahead the others before reaching the global Pareto front, which is a quite small region in a huge objective space. On SPH-m our MOCS approaches are getting better with increasing dimension m and outperform all others.

	NSGA2	SPEA2	PESA2	MOCS-1,2,3		
QV	37	55	51	75	157	45
KUR	37	45	51	74	156	47
SPH-2	44	71	53	74	141	47
SPH-3	45	73	76	47	58	42
SPH-4	46	77	55	87	58	46

Table 4: Average time in minutes on 20 runs on the same processors.

MOCS-2 seems to be the best of our approaches: It uses the NSGA2 Environmental Selection, for Mating Selection it first clusters the population with the x-means technique into the best quantity of clusters then it uses binary tournament region-based selection with the "squeeze"-factor and lastly for every chosen cluster it performs a binary tournament selection with the NSGA2-crowding distance measure. This last binary tournament is absent in PESA1 and PESA2. They use just random selection to choose individuals from a hyperbox, which may explain their bad performance.

Table 3 shows the average runtime in minutes of the algorithms. Just MOCS-2 needs factor 3 more time compared to NSGA2 on some problems. This shows experimentally that the time complexity of MOCS holds $O(mN^2)$ as stated in section 2.3.

5 Conclusion

We described a region-based selection technique in our Multi-Objective Clustering Selection EA, called MOCS. The advantage of MOCS is its automated clustering which clusters the individuals of the population in a very flexible way. In contrast to any hypergrid strategy this prevents choosing the wrong grid size, which leads to too large or too small hyperboxes. Additionally, after the region-based selection step, MOCS uses another binary tournament inside the clusters to increase the selection pressure. Here we used the technique from NSGA2 to select individuals inside a cluster.

MOCS has the same or better performance compared to NSGA2, SPEA2 and PESA2. Furthermore it shows better performance with higher dimensional problems, which we will evaluate on real world problems like the calibration process of combustion engines or in manufacturing industries [Koch et al., 1999, Koch et al., 2001].

Thus we showed that clustering region-based selection with an additional local individual-based selection is a promising alternative to existing methods. For further investigation we want to incorporate other strategies like SPEA2 into our hybrid framework. Also the behaviour of the clustering technique needs to be investigated to get a deeper knowledge of how MOCS works.

References

[Corne et al., 2001] Corne, D. W., Jerram, N. R., Knowles, J. D., and Oates, M. J. (2001). PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization. In Spector, L., Goodman, E. D., Wu, A., Langdon, W., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., and Burke, E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 283–290, San Francisco, California. Morgan Kaufmann Publishers.

[Corne et al., 2000] Corne, D. W., Knowles, J. D., and Oates, M. J. (2000). The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 839–848, Paris, France. Springer. Lecture Notes in Computer Science No. 1917.

[Deb, 2001] Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley.

[Deb et al., 2000] Deb, K., Agrawal, S., Pratab, 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., editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France. Springer. Lecture Notes in Computer Science No. 1917.

[Duda and Hart, 1973] Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley Sons.

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

[Koch et al., 1999] Koch, T. E., Müller, R., Schneider, F., and Zell, A. (1999). Positionierung von Spannmitteln zur Werkstückfixierung mittels Evolutionärer Agorithmen. *Automatisierungstechnische Praxis atp*, pages 32–39.

[Koch et al., 2001] Koch, T. E., Schneider, F., and Zell, A. (2001). Optimal positioning of clamps for workpiece adjustment using multi-objective evolutionary computation. In Parmee, I. C. and Hajela, H. C., editors, *Proceedings of Optimisation in Industries III*, page in press. Springer, London.

[Kursawe, 1991] Kursawe, F. (1991). A variant of evolution strategies for vector optimization. In Schwefel, H. P. and Männer, R., editors, *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 193–197, Berlin, Germany. Springer-Verlag.

[Molyneaux et al., 2001] Molyneaux, A., Leyland, G., and D.Favrat (2001). A New, Clustering Evolutionary Multi-Objective Optimisation Technique. In *Proceedings of the Third International Symposium on Adaptive Systems—Evolutionary Computation and Probabilistic Graphical Models*, pages 41–47, Havana, Cuba. Institute of Cybernetics, Mathematics and Physics.

[Pelleg and Moore, 2000] Pelleg, D. and Moore, A. (2000). X-means: Extending k-means with efficient estimation of the number of clusters. In *Seventeenth International Conference on Machine Learning ICML*, Stanford.

[Quagliarella and Vicini, 1997] Quagliarella, D. and Vicini, A. (1997). Coupling Genetic Algorithms and Gradient Based Optimization Techniques. In Quagliarella, D., Périaux, J., Poloni, C., and Winter, G., editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications*, chapter 14, pages 289–309. John Wiley and Sons, West Sussex, England.

[Schaffer, 1984] Schaffer, J. D. (1984). *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University.

[Schaffer, 1985] Schaffer, J. D. (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum.

[Zitzler et al., 1999] Zitzler, E., Deb, K., and Thiele, L. (1999). Comparison of Multiobjective Evolutionary Algorithms on Test Functions of Different Difficulty. In Wu, A. S., editor, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference. Workshop Program*, pages 121–122, Orlando, Florida.

[Zitzler et al., 2000] Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195.

[Zitzler et al., 2001] Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland.

Evaluation of the Constraint Method-Based Multiobjective Evolutionary Algorithm (CMEA) for a Three-Objective Optimization Problem

Sujay V. Kumar
Hydrological Sciences Branch
NASA Goddard Flight Space Center
Greenbelt, MD 20771.

S. Ranji Ranjithan
Department of Civil Engineering
North Carolina State University
Raleigh, NC 27695.

Abstract

This paper presents a systematic comparative study of CMEA (constraint method-based multiobjective evolutionary algorithm) with several other commonly reported mulitobjective evolutionary algorithms (MOEAs) in solving a three-objective optimization problem. The best estimate of the noninferior space was also obtained by solving this multiobjective (MO) problem using a binary linear programming procedure. Several quantitative metrics are used to compare the noninferior solutions with respect to relative accuracy, as well as spread and distribution of solutions in the noninferior space. Results based on multiple random trials of the MOEAs indicate that overall CMEA performs better than the other MOEAs for this three-objective problem.

1 Introduction

With the recent emergence of interest in solving realistic multiobjective (MO) problems, numerous multiobjective evolutionary algorithms (MOEAs) have been reported in the literature (Deb 2001). While most of them have been successfully tested and evaluated for an array of two-objective test problems, little work is reported on solving MO problems involving more than two objectives. Building upon the study reported by Zitzler et al. (2001) for a three-objective problem, this paper compares and contrasts the performance of the constraint method-based evolutionary algorithm (CMEA) (Ranjithan et al. 2001) with those of SPEA-II (Zitzler et al. 2001), NSGA-II (Deb et al. 2000), and PESA (Corne et al. 2000). These results are also compared with the noninferior set obtained using an MO analysis with a binary linear programming procedure. An array of quantitative metrics is used to conduct a systematic performance comparison among the solutions generated by these MOEAs. In addition to several existing metrics that are extended from the original definitions for two objectives, a new metric is defined to evaluate the relative degree of dominance of one set of noninferior solutions over another.

The next section provides a brief background on CMEA. The subsequent section describes the performance metrics used in this study. Section 4 defines the test problem and a comparison of the results, followed by conclusions.

2 Background

The ϵ-constraint method, typically employed with traditional mathematical programming methods, generates the noninferior set for multiple objectives through iterative solution of the

following single objective problem:

$$\begin{aligned} &\text{Maximize } Z_h(\mathbf{x}) \\ &\text{Subject to } g_i(\mathbf{x}) \leq \forall i = 1, 2, ..., c \\ &\qquad Z_l(\mathbf{x}) \geq Z_l^t \forall l = 1, 2, ..., k; l \neq h \\ &\qquad \mathbf{x} \in X \end{aligned} \tag{1}$$

The problem is assumed to be a maximization problem without loss of generality. Z_h is one of k objectives, Z_l^t is the constraint value for objective $l (l \neq h)$, $\mathbf{x} = \{x_j : j = 1, 2, ..., n\}$ represents the decision vector, X represents the decision space, c is the total number of constraints, and $g_i(\mathbf{x})$ is the i^{th} constraint. The value of Z_l^t is varied incrementally, making the search migrate from one noninferior solution to another.

The evolutionary multiobjective optimization algorithm CMEA combines the ϵ-constraint method for MO within an evolutionary computation framework (Ranjithan et al. 2001). Pareto optimality is achieved in an implicit manner by ensuring the population to migrate along the noninferior surface. A noninferior solution is generated by converging the population to the optimal solution to the above model corresponding to a set of values for Z_l^t. The population is then migrated gradually by incrementally changing the values of Z_l^t. (Please see Ranjithan et al. (2001) for more details.)

A comparison of the results show that the CMEA performs equally or better than SPEA-II, NSGA-II, and PESA for a range of two-objective test problems (Ranjithan et al. 2001, chapter 5 in Kumar 2002). Although the results reported so far have focused on two-objective problems, the underlying concepts and procedures of CMEA are equally applicable to higher order MO problems.

3 Performance Metrics

A spread metric (*Spread*) that determines in each objective space the maximum range represented by the noninferior solutions, and a coverage metric that represents the distribution of the solutions along the noninferior surface were introduced by Ranjithan et al. (2001). Using Figure 1 for illustration, *Spread* in objective Z_1 is the horizontal distance between C_1 and C_q, the two extreme points generated by the MOEA. Similarly, *Spread* in objective Z_2 is the vertical distance between C_1 and C_q. A higher value of the *Spread* metric indicates a better performance.

Two different estimates, $V1$ and $V2$, are defined to characterize the coverage of the noninferior space by the nondominated solutions set (NDS_{MOEA}) generated by an MOEA. Using the notations in Figure 1, $V1$ is defined as $Max\{d_h, \forall h \in \{0, 1, ..., q\}\}$, where d is the distance between adjacent solutions. $V1$ calculations include the set NDS_{MOEA} and the extreme noninferior solutions A and B, which represent the optimum solutions determined separately for each objective. $V2$ is defined as $Max\{d_h, \forall h \in \{1, ..., q-1\}\}$, which includes only the set NDS_{MOEA}. As $V1$ and $V2$ represent the largest gap between adjacent noninferior solutions in the objective space, they characterize how well the solutions generated by an MOEA are distributed to cover the noninferior space. Smaller values for $V1$ and $V2$ indicate a better performance.

Zitzler and Thiele (1999) introduced the C factor that compares two noninferior sets. This parameter can be used to show how the noninferior set of one algorithm dominates the noninferior set of another.

In addition to these metrics, this paper introduces an alternative performance metric called the D (dominance) factor that represents the degree of dominance of noninferior solutions produced by one algorithm over another. For illustration, the two-objective case in Figure 2 shows the noninferior sets NDS_{MOEA-1} and NDS_{MOEA-2} generated by two different MOEAs. To calculate the dominance factor of MOEA-1 over MOEA-2, a distance measure between a solution from the set NDS_{MOEA-1} and the solutions it dominates in the set NDS_{MOEA-2} is determined. In this example, the distance measure for solution i from the set NDS_{MOEA-1} is defined as

Table 1: Summary of Metrics Used for Performance Comparisons in This Paper

Metric	Description	Reference
C factor	Measure of relative dominance of solutions generated by one algorithm over another	Zitzler and Thiele (1999)
$V1$	Measure of the coverage including the known best extreme points	Ranjithan et al. (2001)
$V2$	Measure of the coverage excluding the known best extreme points	Ranjithan et al. (2001)
Spread	Measure of the maximum range covered by the noninferior solutions	Ranjithan et al. (2001)
D factor	Measure of the degree of dominance of solutions generated by one algorithm over another	Figure 2

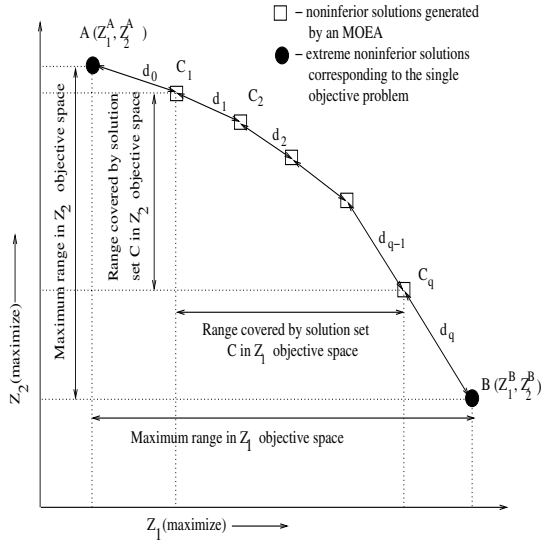


Figure 1: An Example of Two-objective Noninferior Tradeoff to Illustrate the Computation of metrics. d_i represents the distance between two adjacent solutions

$di = \text{Max}\{dij : j = 1, 2, ..., m\}$, where m is the number of solutions it dominates in the set NDS_{MOEA-2}. Then the following aggregate value $D_{1/2}$ is used to define the degree of dominance of MOEA-1 over MOEA-2.

$$D_{1/2} = \frac{\sum_{i=1}^{N} di}{N} \qquad (2)$$

where N is the total number of solutions in the set NDS_{MOEA-1}. The corresponding value for $D_{2/1}$ can be computed similarly.

A summary of the performance metrics used in this paper for the comparison of different algorithms is shown in Table 1. These metrics, although described here for only a two-objective case, are extended for the higher dimensional MO problem presented in this paper.

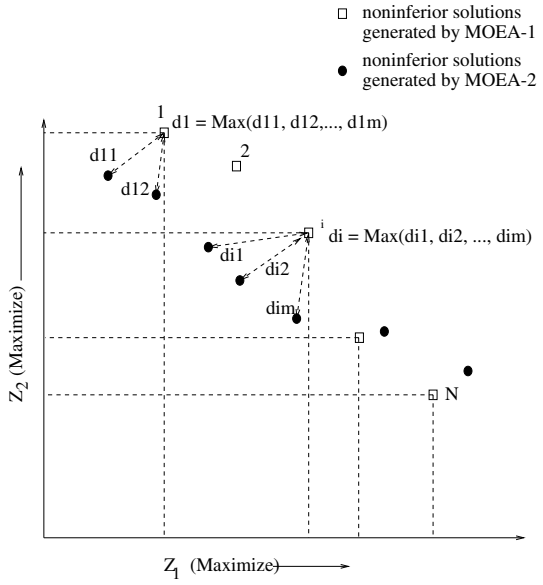


Figure 2: An Example of Two-objective Noninferior Tradeoff to Illustrate the Computation of D factor

4 Testing of CMEA for a Three-Objective Problem

The extended 0/1 multiobjective knapsack problem presented by Zitzler and Thiele (1999) is a constrained, binary, combinatorial search problem. This MO knapsack problem extends a single objective problem by incorporating a number of knapsacks that can be filled by items selected from a common collection of items. The goal is to choose the allocation of items in different knapsacks so that the payoff of each knapsack is maximized without violating the respective weight capacity constraint. This problem is defined mathematically as follows:

$$\begin{aligned} &\text{Maximize } Z_l(\mathbf{x}) = \sum_{j=1}^{n} p_{l,j} x_j \ \forall l = 1, 2, ..., k \\ &\text{Subject to } \sum_{j=1}^{n} w_{l,j} x_j \leq c_l \ \forall l = 1, 2, ..., k \\ &\qquad x_j \in \{0, 1\} \end{aligned} \tag{3}$$

where, $Z_l(\mathbf{x})$ is the total profit associated with the knapsack l, $p_{l,j}$ is the profit of placing item j in knapsack l, $w_{l,j}$ is the weight of item j when placed in knapsack l, c_l is the capacity of knapsack l, $\mathbf{x} = (x_1, x_2, ..., x_n) \in \{0, 1\}^n$ such that $x_j = 1$ if selected and $= 0$ otherwise, n is the total number of available items, and k is the total number of knapsacks.

In this paper, an instance of this multiobjective knapsack problem with three knapsacks ($k = 3$), each with 750 items ($n = 750$) is considered. In addition to solving this problem using CMEA, a binary linear programming solver (BLP), CPLEX®, was also used to generate a set of noninferior solutions by solving Model (1) iteratively. The problem was solved using CMEA (with a population of 100, binary tournament selection, uniform crossover, adaptive mutation, and 289 intervals in each objective) for 10 different random seeds, and the performance metrics listed in Table 1 are calculated based on these 10 runs. The results obtained using CMEA is compared with those obtained using SPEA-II, NSGA-II, and PESA. The performance metrics for SPEA-II, NSGA-II, and PESA are calculated based on 30 different sets of solutions reported by Zitzler et al. (2001).

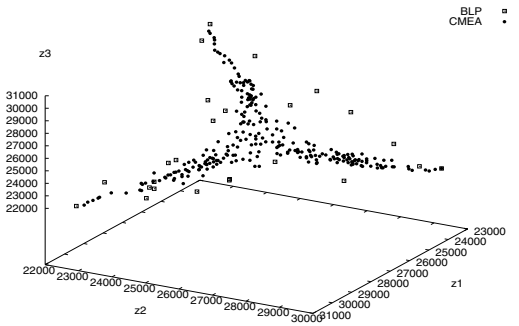


Figure 3: A Comparison of the Noninferior Sets Obtained Using CMEA and BLP

Figure 3 compares the noninferior solutions generated by CMEA and the BLP solutions. The CMEA solutions appear to represent most of the noninferior surface defined by the BLP solutions, which represent the best estimate of the noninferior front available for this problem. The extreme regions, associated with the three "tail-like" sections, are well represented by the CMEA solutions. Figures 4 to 6 compare the noninferior solutions generated by SPEA-II, NSGA-II, and PESA, respectively, with those generated by CMEA. The results from SPEA-II, NSGA-II, and PESA required approximately 576,000 function evaluations (Zitzler et al. 2001), and the CMEA solutions are based on approximately 594,000 function evaluations. While the solutions by SPEA-II, NSGA-II, and PESA solutions well represent the "center" region of the noninferior surface, they entirely miss the three tail regions.

The performance metrics listed in Table 1 were computed for the solutions generated by the MOEAs, and are compared in Figures 7 to 15. These graphs show for each metric the average and the range based on different random trials. The *Spread* metrics in the three objective

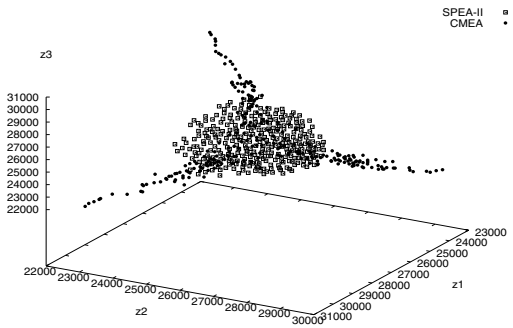


Figure 4: A Comparison of the Noninferior Sets Obtained Using CMEA and SPEA-II

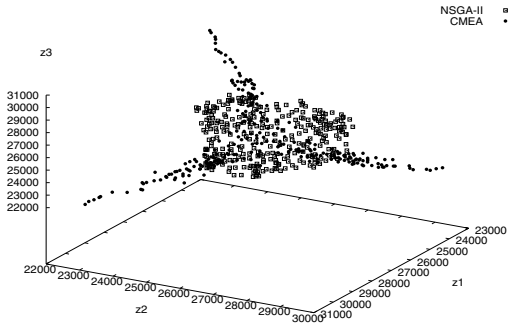


Figure 5: A Comparison of the Noninferior Sets Obtained Using CMEA and NSGA-II

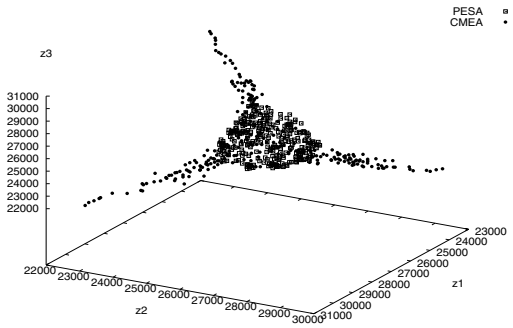


Figure 6: A Comparison of the Noninferior Sets Obtained Using CMEA and PESA

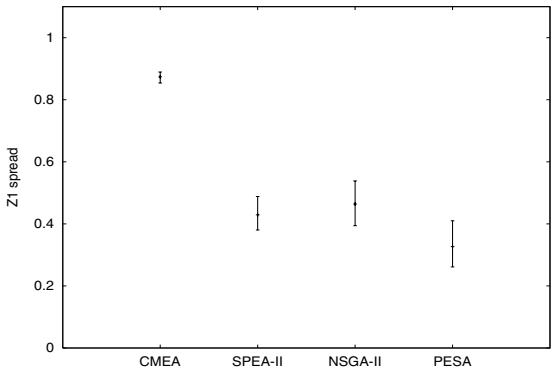


Figure 7: Z_1 spread comparison for CMEA, SPEA-II, NSGA-II, and PESA (a higher value indicates a better performance)

spaces are shown in Figures 7 to 9. These figures indicate that CMEA outperforms the other algorithms compared in this study. Figure 10 compares the coverage metric $V2$. While all MOEAs perform similarly, PESA shows a slight edge over the others. When comparing metric $V1$ in Figure 11, CMEA clearly outperforms the other MOEAs. This confirms the observations (from Figures 4 to 6) that SPEA-II, NSGA-II, and PESA solutions cover only the central region of the noninferior surface while the CMEA solutions are broadly distributed.

Comparisons of C factors are shown in Figures 12 and 13. This further confirms the conclusions drawn above about coverage based on the $V1$ and $V2$ metrics. Comparisons of D-factor are shown in Figures 14 and 15. By collectively interpreting the values of $D_{\text{CMEA/another MOEA}}$ and $D_{\text{another MOEA/CMEA}}$ for each MOEA compared, it can be concluded that CMEA solutions dominate those of the other MOEAs to a higher degree than *vice versa*. Alternatively, it implies that when CMEA solutions dominate solutions of another MOEA, they dominate them to a higher degree than when solutions of another MOEA dominate solutions of CMEA.

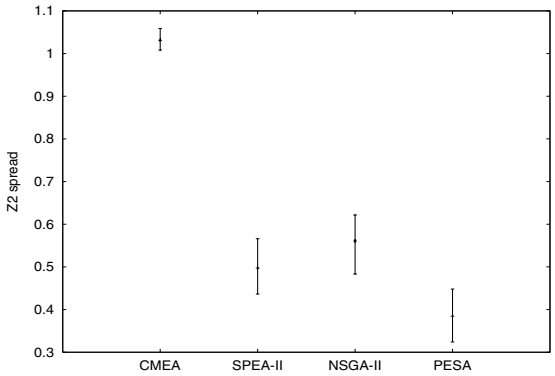


Figure 8: Z_2 spread comparison for CMEA, SPEA-II, NSGA-II, and PESA (a higher value indicates a better performance)

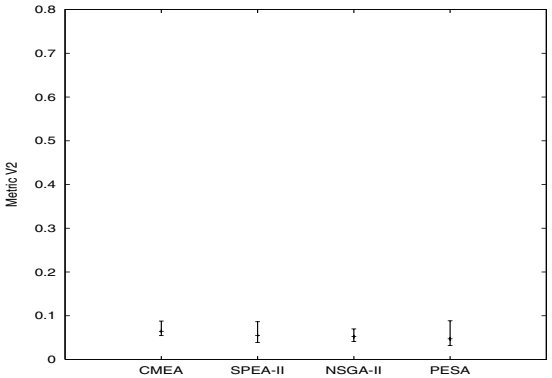


Figure 10: Coverage metric $V2$ comparison for CMEA, SPEA-II, NSGA-II, and PESA (a lower value indicates a better performance)

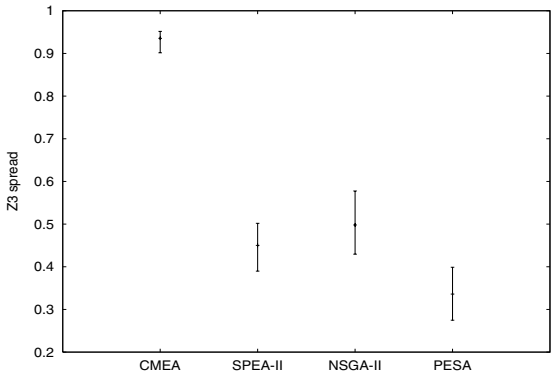


Figure 9: Z_3 spread comparison for CMEA, SPEA-II, NSGA-II, and PESA (a higher value indicates a better performance)

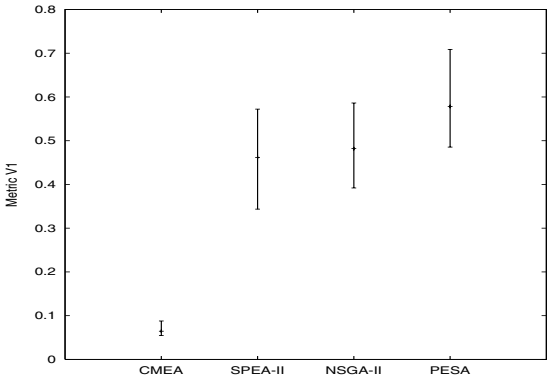


Figure 11: Coverage metric $V1$ comparison for CMEA, SPEA-II, NSGA-II, and PESA (a lower value indicates a better performance)

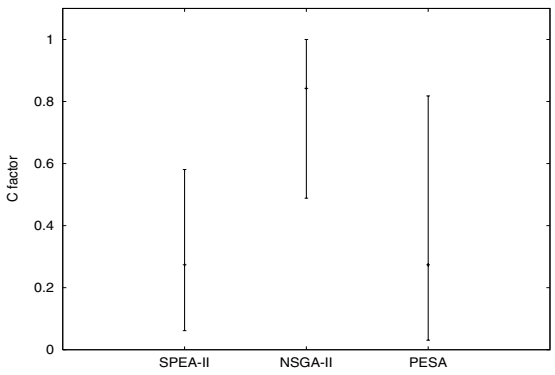


Figure 12: Comparison of C factor: CMEA solutions over SPEA-II, NSGA-II, and PESA solutions

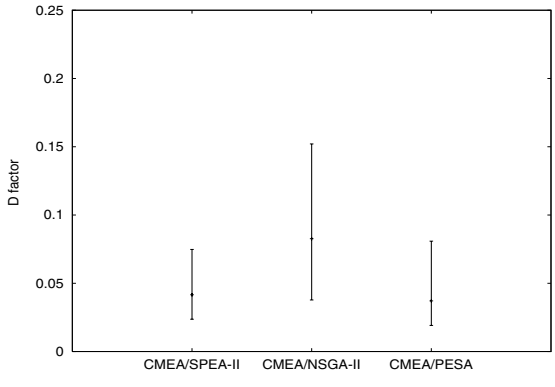


Figure 14: Comparison of D factor: CMEA solutions over SPEA-II, NSGA-II, and PESA solutions (a larger value indicates a higher degree of dominance of CMEA over another MOEA)

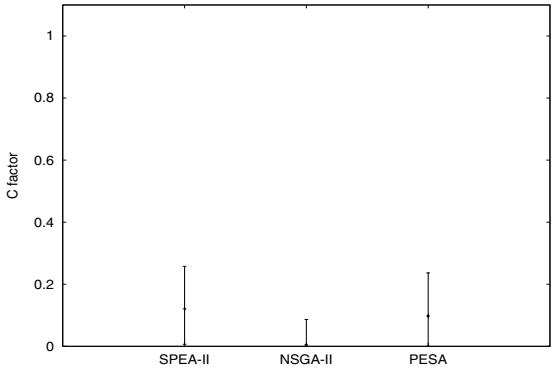


Figure 13: Comparison of C factor: SPEA-II, NSGA-II, and PESA solutions over CMEA solutions

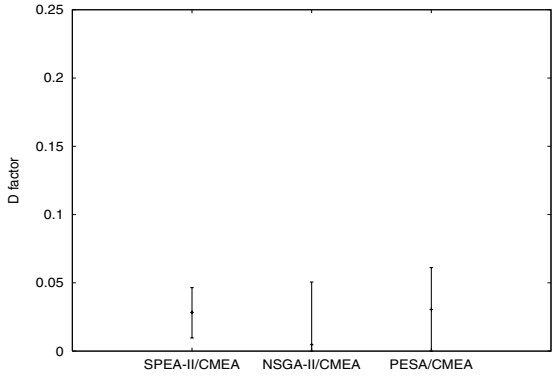


Figure 15: Comparison of D factor: SPEA-II, NSGA-II, and PESA solutions over CMEA solutions (a larger value indicates a higher degree of dominance of another MOEA over CMEA)

5 Conclusions

Comparing the noninferior solutions obtained using SPEA-II, NSGA-II, and PESA with those obtained using BLP (which is the best available estimate of the noninferior surface), CMEA performs relatively well in finding noninferior solutions that are close to the best available estimation, as well as in covering most of the noninferior surface for the thee-objective extended knapsack problem. When comparing the solutions obtained by the different MOEAs tested in this study, CMEA performs better than all others with respect to the spread of solutions in the noninferior space. While CMEA solutions are able to cover a broader portion of the noninferior surface, the other MOEAs generate a high density of solutions in the central portion of the noninferior surface. In the context of accuracy or degree of dominance, the CMEA solutions dominate those generated by the other MOEAs relatively more frequently.

While this study provides a systematic comparison of several MOEAs for only one three-objective optimization problem, further testing and evaluation studies are needed. Similar to the large array of test problems used in two-objective MO optimization, additional three-objective test problems reflecting different problem complexities need to be defined and be used in further comparative studies of these MOEAs. Also, the scale-up implications of higher number of objectives on the computational needs of the different MOEAs need to be investigated.

6 Acknowledgments

The authors would like to thank Dr. Eckart Zitzler for sharing their results obtained using NSGA-II, SPEA-II, and PESA for the test problem reported in this paper.

References

Corne, D. W., J. D. Knowles, and M. J. Oates (2000). The pareto-envelope based selection algorithm for multiobjective optimixation. In S. et al. (Ed.), *Parallel Problem Solving from Nature - PPSN VI*, Lecture Notes in Computer Science, pp. 869–878. Springer Verlag.

Deb, K. (2001). *Multi-objective Optimization Using Evolutionary Algorithms*. England: John Wiley and Sons, Ltd.

Deb, K., A. Pratap, and T. Meyarivan (2000). A fast elitist non-dominated sorting genetic algorithm for multiobjective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI*, pp. 849–858.

Kumar, S. V. (2002). *Vitri - A Generic Framework for Engineering Decision Support on Heterogeneous Computer Networks*. Ph. D. thesis, North Carolina State University.

Ranjithan, S. R., S. K. Chetan, and H. K. Dakshina (2001). Constraint method-based evolutionary algorithm (CMEA) for multiobjective optimization. In E. Z. et al. (Ed.), *Evolutionary Multi-Criteria Optimization 2001*, Lecture Notes in Computer Science 1993, pp. 299–313. Springer-Verlag.

Zitzler, E., M. Laumanns, and L. Thiele (2001). Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Communications Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Gloriastrasse 35, CH-8092.

Zitzler, E. and L. Thiele (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 2*(4), 257–271.

Archiving with Guaranteed Convergence and Diversity in Multi-Objective Optimization

Marco Laumanns, Lothar Thiele, Eckart Zitzler
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH) Zurich
CH-8092 Zurich, Switzerland
{laumanns,thiele,zitzler}@tik.ee.ethz.ch

Kalyanmoy Deb
Department of Mechanical Engineering
Indian Institute of Technology Kanpur
Kanpur, PIN 208 016, India
deb@iitk.ac.in

Abstract

Over the past few years, the research on evolutionary algorithms has demonstrated their niche in solving multi-objective optimization problems, where the goal is to find a number of Pareto-optimal solutions in a single simulation run. However, none of the multi-objective evolutionary algorithms (MOEAs) has a proof of convergence to the true Pareto-optimal solutions with a wide diversity among the solutions. In this paper we discuss why a number of earlier MOEAs do not have such properties. A new archiving strategy is proposed that maintains a subset of the generated solutions. It guarantees convergence and diversity according to well-defined criteria, i.e. ϵ-dominance and ϵ-Pareto optimality.

1 Introduction

After the doctoral study of Schaffer (1984) on the vector evaluated genetic algorithm (VEGA), Goldberg's suggestion of the use of non-dominated sorting along with a niching mechanism (1989) generated an overwhelming interest on multi-objective evolutionary algorithms (MOEAs). Initial MOEAs – MOGA (Fonseca and Fleming 1993), NSGA (Srinivas and Deb 1994), NPGA (Horn et al. 1994) – used Goldberg's suggestion in a straightforward manner: (i) the fitness of a solution was assigned using the extent of its *domination* in the population and (ii) the diversity among solutions was preserved using a *niching* strategy. The above three studies have shown that different ways of implementing the above two tasks can all result in successful MOEAs. However, in order to ensure convergence to the true Pareto-optimal solutions, an elite-preservation operator was absent in those algorithms. Thus, the latter MOEAs mainly concentrated on how elitism could be introduced in an MOEA. This resulted in a number of advanced algorithms – SPEA (Zitzler and Thiele 1999), PAES (Knowles and Corne 2000), NSGA-II (Deb et al. 2000), and others. With the development of better algorithms, multi-objective evolutionary algorithms have also been used in a number of application case studies (Zitzler et al. 2001).

What is severely lacking are studies related to theoretical convergence analysis with guaranteed spread of solutions. In this regard, Rudolph (1998, 2001) and Rudolph and Agapie (2000) suggested a series of algorithms, all of which guarantee convergence, but do not address the following two aspects:

1. The convergent algorithms do not guarantee maintaining a spread of solutions.
2. The algorithms do not specify any time complexity for their convergence to the true Pareto-optimal set.

Although the second task is difficult to achieve (and is dependent on the fitness landscape and genetic operators used) even in the case of single-objective evolutionary algorithms, the first task is as important as the task of converging to the true Pareto set. Deb (2001) suggested a steady-state MOEA that attempts to maintain spread while attempting to converge to the true Pareto-optimal front, but there is no proof for its convergence properties. Knowles (2002) has analyzed two further possibilities, metric-based archiving and adaptive grid archiving. The metric-based strategy requires a function that assigns a scalar value to each possible approximation set reflecting its quality and fulfilling certain monotony conditions. Convergence then is proven as a local optimum of the quality function will be reached, but how this optimum relates to the actual distribution of the solutions is unclear and the computational overhead is enormous. The adaptive grid archiving strategy implemented in PAES provably maintains solutions in some 'critical' regions of the Pareto set once they have been found, but convergence can only be guaranteed for the solutions at the extremes of the Pareto set.

Algorithm 1 Iterative search procedure

1: $t := 0$
2: $A^{(0)} := \emptyset$
3: **while** $terminate(A^{(t)}, t) = false$ **do**
4: $\quad t := t + 1$
5: $\quad f^{(t)} := generate()$ {generates new search point}
6: $\quad A^{(t)} := update(A^{(t-1)}, f^{(t)})$ {updates archive}
7: **end while**
8: **Output:** $A^{(t)}$

In this paper, we propose an archiving/selection strategy that guarantees at the same time progress towards the Pareto-optimal set and a covering of the whole range of the non-dominated solutions. The algorithm maintains a finite-sized archive of non-dominated solutions which gets iteratively updated in the presence of a new solution based on the concept of ϵ-dominance. The use of ϵ-dominance also makes the algorithms practical by allowing a decision-maker to control the resolution of the Pareto set approximation by choosing an appropriate ϵ value.

In the remainder of the paper, we state the general structure of an iterative archive-based search procedure which is usually used for multi-objective optimization. In section 3 we formally define our concepts of ϵ-dominance and ϵ-Pareto optimality. We present the new archiving algorithm and prove the required convergence and distribution properties. The simulation results in section 4 illustrate its practical relevance in contrast to existing algorithms which either fail with respect to convergence or to distribution behavior.

2 Structure of an Iterative Multi-Objective Search Algorithm

The purpose of this section is to informally describe the problem we are dealing with. To this end, let us first give a template for a large class of iterative search procedures which are characterized by the generation of a sequence of search points and a finite memory.

The purpose of such algorithms is to find or approximate the Pareto set of the image set F of a vector valued function $h : X \rightarrow F$ defined over some domain X. In the context of multi-objective optimization, h, F and X are often called the multi-valued objective function, the objective space and the decision space, respectively.

An abstract description of a generic iterative search algorithm is given in Algorithm 1. The integer t denotes the iteration count, the n-dimensional vector $f^{(t)} \in F$ is the sample generated at iteration t and the set $A^{(t)}$ will be called the archive at iteration t and should contain a representative subset of the samples in the objective space F generated so far. To simplify the notation, we represent

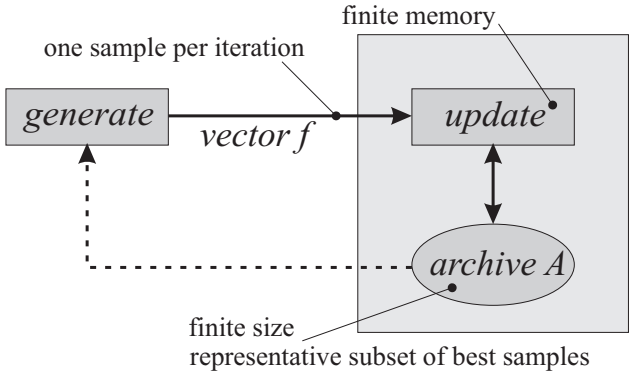


Figure 1: Representation of the generic search algorithm 1.

samples by n-dimensional real vectors f where each coordinate represents one of the objective values. Additional information about the corresponding decision values could be associated to f, but will be of no concern in this paper.

The purpose of the function *generate* is to generate a new solution in each iteration t, possibly using the contents of the old archive set $A^{(t-1)}$. The function *update* gets the new solution $f^{(t)}$ and the old archive set $A^{(t-1)}$ and determines the updated one, namely $A^{(t)}$. In general, the purpose of this sample storage is to gather 'useful' information about the underlying search problem during the run. Its use is usually two-fold: On the one hand it is used to store the 'best' solutions found so far, on the other hand the search operator exploits this information to steer the search to promising regions.

This algorithm could easily be viewed as an evolutionary algorithm when the $generate$ operator is associated with variation (recombination and mutation). However, we would like to point out that all following investigations are equally valid for any kind of iterative process which can be described as Algorithm 1 and used for approximating the Pareto set of multi-objective optimization problems, e.g. simulated annealing or tabu search.

There are several reasons, why the archive $A^{(t)}$ should be of constant size, independent of the number of iterations t. At first, the computation time grows with the number of archived solutions, as for example the function *generate* may use it for guiding the search, or it may simply be impossible to store all solutions as the physical memory is always finite. In addition, the value of presenting such a large set of solutions to a decision maker is doubtful in the context of decision support, instead one should provide him with a set of the best *representative* samples. Finally, in limiting the solution set preference information could be used to steer the process to certain parts of the search space.

The paper solely deals with the function *update*, i.e. with an

appropriate generation of the archive. Because of the reasons described above, the corresponding algorithm should have the following properties, see also Fig. 1:

- The algorithm is provided with one sample $f^{(t)}$ at each iteration, i.e. one at a time.
- It operates with finite memory. In particular, it cannot store all the samples submitted until iteration t.
- The algorithm should maintain a set $A^{(t)}$ of a limited size which is independent of the iteration count t. The set should contain a representative subset of the best samples $f^{(1)}, ..., f^{(t)}$ received so far.

A clear definition of the term *representative subset* of the *best samples* will be given in Section 3.1. But according to the common notion of optimality in multi-objective optimization and the above discussion it should be apparent that the archive $A^{(t)}$ should contain a subset of all Pareto vectors of the samples generated until iteration t. In addition, these selected Pareto vectors should represent the diversity of all Pareto vectors generated so far. Such an algorithm in will be constructed in section 3.3.

3 Algorithms for Convergence and Diversity

Before we present the update functions for finding a diverse set of Pareto-optimal solutions, we define some terminology.

3.1 Concept of Pareto Set Approximation

In this section we define relevant concepts of dominance and (approximate) Pareto sets. Without loss of generality, we assume a normalized and positive objective space in the following for notational convenience. The algorithms presented in this paper assume that all objectives are to be maximized. However, either by using the *duality* principle (Deb 2001) or by simple modifications to the domination definitions, these algorithms can be used to handle minimization or combined minimization and maximization problems.

Objective vectors are compared according to the dominance relation defined below and displayed in Fig. 2 (left).

Definition 1 (Dominance relation)
Let $f, g \in \mathbb{R}^m$. Then f is said to dominate g, denoted as $f \succ g$, iff

1. $\forall i \in \{1, \ldots, m\} : f_i \geq g_i$
2. $\exists j \in \{1, \ldots, m\} : f_j > g_j$

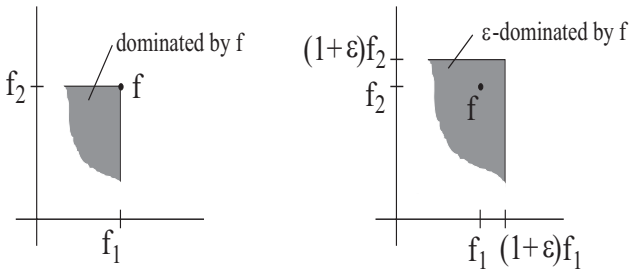


Figure 2: Graphs visualizing the concepts of dominance (left) and ϵ-dominance (right).

Definition 2 (Pareto set)
Let $F \subseteq \mathbb{R}^m$ be a set of vectors. Then the Pareto set F^* of F is defined as follows: F^* contains all vectors $g \in F$ which are not dominated by any vector $f \in F$, i.e.

$$F^* := \{g \in F | \ \nexists f \in F : f \succ g\} \tag{1}$$

Vectors in F^* are called Pareto vectors of F. The set of all Pareto sets of F is denoted as $P^*(F)$.

From the above definition we can easily deduce that any vector $g \in F \setminus F^*$ is dominated by at least one $f \in F^*$, i.e.

$$\forall g \in F \setminus F^* : \quad \exists f \in F^* \quad \text{such that} \quad f \succ g. \tag{2}$$

Moreover, for a given set F, the set F^* is unique. Therefore, we have $P^*(F) = \{F^*\}$. For many sets F, the Pareto set F^* is of substantial size. Thus, the numerical determination of F^* is prohibitive, and F^* as a result of an optimization is questionable. Moreover, it is not clear at all what a decision maker can do with such a large result of an optimization run. What would be more desirable is an approximation of F^* which *approximately* dominates all elements of F and is of (polynomially) bounded size. This set can then be used by a decision maker to determine interesting regions of the decision and objective space which can be explored in further optimization runs. Next, we define a generalization of the dominance relation as visualized in Fig. 2 (right).

Definition 3 (ϵ-Dominance)
Let $f, g \in \mathbb{R}^{+m}$. Then f is said to ϵ-dominate g for some $\epsilon > 0$, denoted as $f \succ_\epsilon g$, iff for all $i \in \{1, \ldots, m\}$

$$(1 + \epsilon) \cdot f_i \geq g_i. \tag{3}$$

Definition 4 (ϵ-approximate Pareto set)
Let $F \subseteq \mathbb{R}^{+m}$ be a set of vectors and $\epsilon > 0$. Then a set F_ϵ is called an ϵ-approximate Pareto set of F, if any vector $g \in F$ is ϵ-dominated by at least one vector $f \in F_\epsilon$, i.e.

$$\forall g \in F : \exists f \in F_\epsilon \quad \text{such that} \quad f \succ_\epsilon g. \tag{4}$$

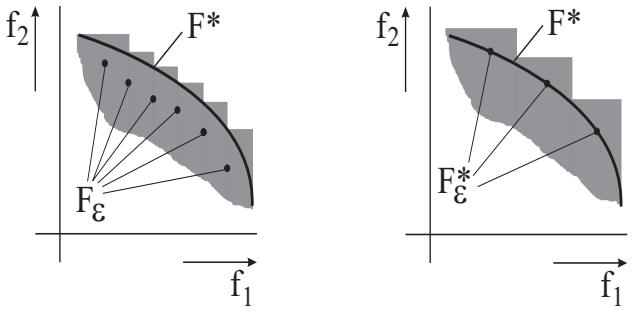


Figure 3: Graphs visualizing the concepts of ϵ-approximate Pareto set (left) and ϵ-Pareto set (right).

The set of all ϵ-approximate Pareto sets of F is denoted as $P_\epsilon(F)$.

Of course, the set F_ϵ is not unique. Many different concepts for ϵ-efficiency[1] and the corresponding Pareto set approximations exist in the operations research literature, a survey is given by Helbig and Pateva (1994). As most of the concepts deal with infinite sets, they are not practical for our purpose of producing and maintaining a representative subset. Nevertheless they are of theoretical interest and have nice properties which can for instance be used in convergence proofs, see (Hanne 1999) for an application in MOEAs.

Using discrete ϵ-approximations of the Pareto set was suggested simultaneously by Evtushenko and Potapov (1987), Reuter (1990), and Ruhe and Fruhwirt (1990). As in our approach, each Pareto-optimal point is approximately dominated by some point of the representative set. The first two papers use absolute deviation (additive ϵ, see Eqn. 7 below) and the third relative deviation (multiplicative ϵ as above), but they are not concerned with the size of the representative set in the general case.

Recently, Papadimitriou and Yannakakis (2000) and Erlebach et al. (2001) have pointed out that under certain assumptions there is always an approximate Pareto set whose size is polynomial in the length of the encoded input. This can be achieved by placing a hyper-grid in the objective space using the coordinates $1, (1+\epsilon), (1+\epsilon)^2, \ldots$ for each objective. As it suffices to have one representative solution in each grid cell and to have only non-dominated cells occupied, it can be seen that for any finite ϵ and any set F with bounded vectors f, i.e. $1 \leq f_i \leq K$ for all $i \in \{1, \ldots, m\}$, there exists a set F_ϵ containing

$$|F_\epsilon| \leq \left(\frac{\log K}{\log (1+\epsilon)} \right)^{m-1} \tag{5}$$

vectors. A proof will be given in connection with Alg. 2 in section 3.3.

Note that the concept of approximation can also be used if other similar definitions of ϵ-dominance are used, e.g. the following additive approximation

$$\epsilon_i + f_i \geq g_i \quad \forall i \in \{1, \ldots, m\} \tag{6}$$

where ϵ_i are constants, separately defined for each coordinate. In this case there exist ϵ-approximate Pareto sets whose size can be bounded as follows:

$$|F_\epsilon| \leq \prod_{j=1}^{m-1} \frac{K-1}{\epsilon_i} \tag{7}$$

where $1 \leq f_i \leq K$, $K \geq \epsilon_i$ for all $i \in \{1, \ldots, m\}$. A further refinement of the concept of ϵ-approximate Pareto sets leads to the following definition.

Definition 5 (ϵ-Pareto set)
Let $F \subseteq \mathbb{R}^{+m}$ be a set of vectors and $\epsilon > 0$. Then a set $F_\epsilon^* \subseteq F$ is called an ϵ-Pareto set of F if

1. *F_ϵ^* is an ϵ-approximate Pareto set of F, i.e. $F_\epsilon^* \in P_\epsilon(F)$, and*
2. *F_ϵ^* contains Pareto points of F only, i.e. $F_\epsilon^* \subseteq F^*$.*

The set of all ϵ-Pareto sets of F is denoted as $P_\epsilon^*(F)$.

The above defined concepts are visualized in Fig. 3. An ϵ-Pareto set F_ϵ^* not only ϵ-dominates all vectors in F, but also consists of Pareto-optimal vectors of F only, therefore we have $P_\epsilon^*(F) \subseteq P_\epsilon(F)$.

Since finding the Pareto set of an arbitrary set F is usually not practical because of its size, one needs to be less ambitious in general. Therefore, the ϵ-approximate Pareto set is a practical solution concept as it not only represents all vectors F but also consists of a smaller number of elements. Of course, an ϵ-Pareto set is more attractive as it consists of Pareto vectors only.

3.2 Convergence and Diversity

Convergence and diversity can be defined in various ways. Here, we consider the objective space only. According to Definition 3, the ϵ value stands for a relative "tolerance" that we allow for the objective values. In contrast, using equation (6) we would allow a constant additive (absolute) tolerance.

[1] The terms "efficient" and "Pareto-optimal" can be used synonymously. While the former appears to be more frequent in operations research literature, we generally use the latter as it is more common the field of evolutionary computation.

Algorithm 2 $update$ function for ϵ-Pareto set

1: **Input:** A, f
2: $D := \{f' \in A | box(f) \succ box(f')\}$
3: **if** $D \neq \emptyset$ **then**
4: $\quad A' := A \cup \{f\} \setminus D$
5: **else if** $\exists f' : (box(f') = box(f) \wedge f \succ f')$ **then**
6: $\quad A' := A \cup \{f\} \setminus \{f'\}$
7: **else if** $\nexists f' : box(f') = box(f) \vee box(f') \succ box(f)$ **then**
8: $\quad A' := A \cup \{f\}$
9: **else**
10: $\quad A' := A$
11: **end if**
12: **Output:** A'

Algorithm 3 function box

1: **Input:** f
2: **for all** $i \in \{1, \ldots, m\}$ **do**
3: $\quad b_i := \lfloor \frac{\log f_i}{\log (1+\epsilon)} \rfloor$
4: **end for**
5: $b := (b_1, \ldots, b_m)$
6: **Output:** b {box index vector}

The choice of the ϵ value is application specific: A decision maker should choose a type and magnitude that suits the (physical) meaning of the objective values best. The ϵ value further determines the maximal size of the archive according to equations (5) and (7).

3.3 Maintaining an ϵ-Pareto Set

The Algorithm 2 has a two level concept. On the coarse level, the search space is discretized by a division into boxes (see Algorithm 3), where each vector uniquely belongs to one box. Using a generalized dominance relation on these boxes, the algorithm always maintains a set of non-dominated boxes, thus guaranteeing the ϵ-approximation property. On the fine level at most one element is kept in each box. Within a box, each representative vector can only be replaced by a dominating one (similar to Agapie's and Rudolph's algorithm), thus guaranteeing convergence.

Now, we can prove the convergence of the above update strategy to the Pareto set while preserving diversity of solution vectors at the same time.

Theorem 1
Let $F^{(t)} = \bigcup_{j=1}^{t} f^{(j)}, 1 \leq f_i^{(j)} \leq K$, be the set of all vectors created in Algorithm 1 and given to the $update$ function as defined in Algorithm 2. Then $A^{(t)}$ is an ϵ-Pareto set of $F^{(t)}$ with bounded size according to Eq. (5), i.e.

1. $A^{(t)} \in P_\epsilon^*(F^{(t)})$
2. $|A^{(t)}| \leq \left(\frac{\log K}{\log (1+\epsilon)}\right)^{(m-1)}$

Proof.

1. Suppose the algorithm is not correct, i.e. $A^{(t)} \notin P_\epsilon^*(F^{(t)})$ for some t. According to Def. 5 this occurs only if some $f = f^{(\tau)}, \tau \leq t$ is (1) not ϵ-dominated by any member of $A^{(t)}$ and not in $A^{(t)}$ or (2) in $A^{(t)}$ but not in the Pareto set of $F^{(t)}$.

   Case (1): For $f = f^{(\tau)}$ not being in $A^{(t)}$, it can either have been rejected at $t = \tau$ or accepted at $t = \tau$ and removed later on. Removal, however, only takes place when some new f' enters A, which dominates f (line 6) or whose box value dominates that of f (line 4). Since both relations are transitive, and since they both imply ϵ-dominance, there will always be an element in A which ϵ-dominates f, which contradicts the assumption. On the other hand, f will only be rejected if there is another $f' \in A^{(\tau)}$ with the same box value and which is not dominated by f (line 10). This f', in turn, ϵ-dominates f and – with the same argument as before – can only be replaced by accepting elements which also ϵ-dominate f.

   Case (2): Since f is not in the Pareto set of $F^{(t)}$, there exists $f' = f^{(\tau')}, \tau' \neq \tau, f' \in F^{*(t)}$ with $f' \succ f$. This implies $box(f') \succ box(f)$ or $box(f') = box(f)$. Hence, if $\tau' < \tau$, f would not have been accepted. If $\tau' > \tau$, f would have been removed from A. Thus, $f \notin A^{(t)}$, which contradicts the assumption.

2. The objective space is divided into $\left(\frac{\log K}{\log (1+\epsilon)}\right)^m$ boxes, and from each box at most one point can be in $A^{(t)}$ at the same time. Now consider the $\left(\frac{\log K}{\log (1+\epsilon)}\right)^{(m-1)}$ equivalence classes of boxes where – without loss of generality – in each class the boxes have the same coordinates in all but one dimension. There are $\frac{\log K}{\log (1+\epsilon)}$ different boxes in each class constituting a chain of dominating boxes. Hence, only one point from each of these classes can be a member of $A^{(t)}$ at the same time.

□

As a result, Algorithm 2 uses a finite memory, successively updates a finite subset of vectors that ϵ-dominate all vectors generated so far. It can be guaranteed that the subset contains only elements which are not dominated by any of the generated vectors. Note that specific bounds on the objective values are *not* used in the algorithm itself and are not required for the convergence proof (claim 1 of Theorem 1).

They are only utilized to prove the relation between ϵ and the size of the archive given in the second claim.

4 Simulations

This section presents some simulation results to demonstrate the behavior of the proposed algorithm for two example multi-objective optimization problems (MOPs). We use instances of the iterative search procedure (specified in Alg. 1) with a common generator and examine different *update* operators.

An isolated assessment of the update strategy of course requires the generator to act independently from the archive set $A^{(t)}$ to guarantee that exactly the same sequence of points is given to the update function for all different strategies. Despite that, the exact implementation of the generator is irrelevant for this study, therefore we use standard MOEAs here and take the points in the sequence of their generation as input for the different update functions.

4.1 Convergence Behavior

At first we are interested in how different update strategies affect the *convergence* of the sequence $(A^{(t)})$. As a test problem a two-objective knapsack problem with 100 items is taken from (Zitzler and Thiele 1999). The low number of decision variables is sufficient to show the anticipated effects, and we found it advantageous for visualization and comparison purposes to be able to compute the complete Pareto set F^* beforehand via Integer Linear Programming.

The points given to update operator are generated by a standard NSGA-II with population size 100, one-point crossover, and bit-flip mutations (with probability $4/n = 0.04$). Figure 4 shows the output $A^{(t)}$ of sample runs for the different instances after $t = 5,000,000$ and $t = 10,000,000$ iterations (generated objective vectors), using update operators from SPEA, NSGA-II (both with maximum archive size of 20 and Alg. 2 with $\epsilon = 0.01$.

It is clearly visible that both the archiving (selection) strategies from SPEA and NSGA-II suffer from the problem of partial deterioration: Non-dominated points – even those which belong to the "real" Pareto set – can get lost, and on the long run might even be replaced by dominated solutions. This is certainly not desirable, and algorithms relying on these strategies cannot be claimed to be convergent, even if the generator would be able to produce all elements of the Pareto set F^*[2]. In contrast, Alg. 2 is able to maintain an ϵ-Pareto set of the generated solutions over time.

[2]In our experiments almost all Pareto-optimal points have been produced by the generator within $t = 10,000,000$ iterations.

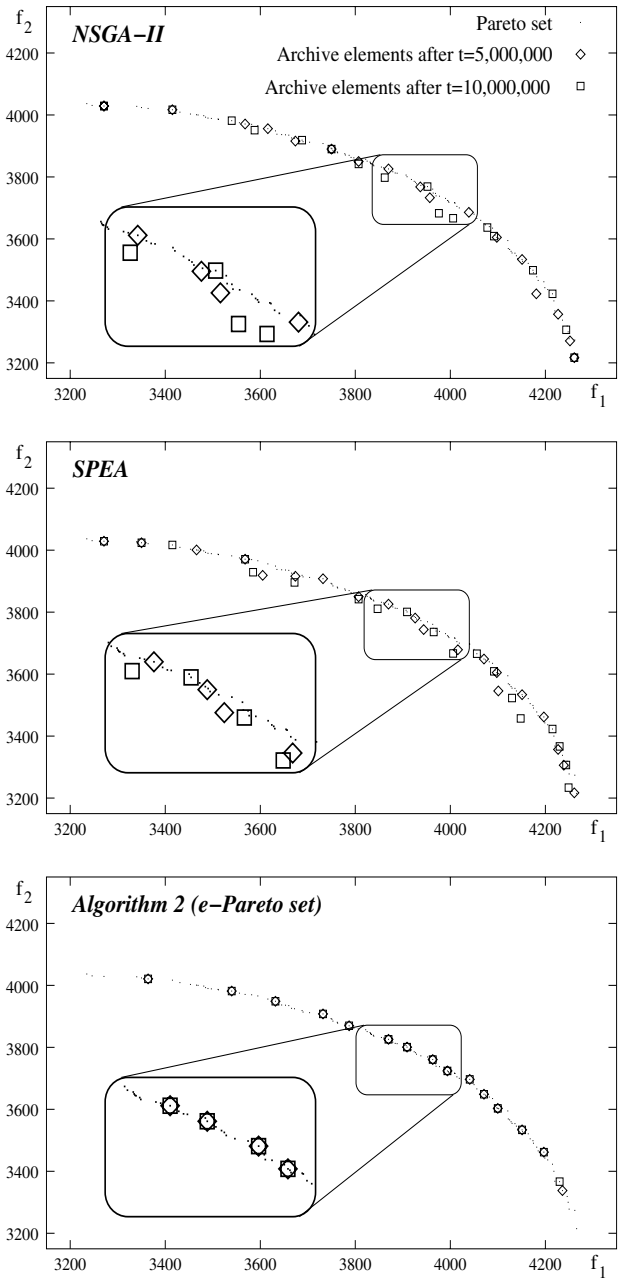


Figure 4: Objective space of the knapsack problem, the dots show the elements of the Pareto set F^*. The different figures correspond to different instances of the *update* operator in Alg. 1: NSGA-II (upper left), SPEA (upper right), and Alg. 2 (lower row). In each figure the archive set $A^{(t)}$ is shown, for $t = 5,000,000$ (with diamonds) and for $t = 10,000,000$ (with boxes). A subset of the samples is highlighted so visualize the negative effect of losing Pareto-optimal solutions in many current archiving/selection schemes.

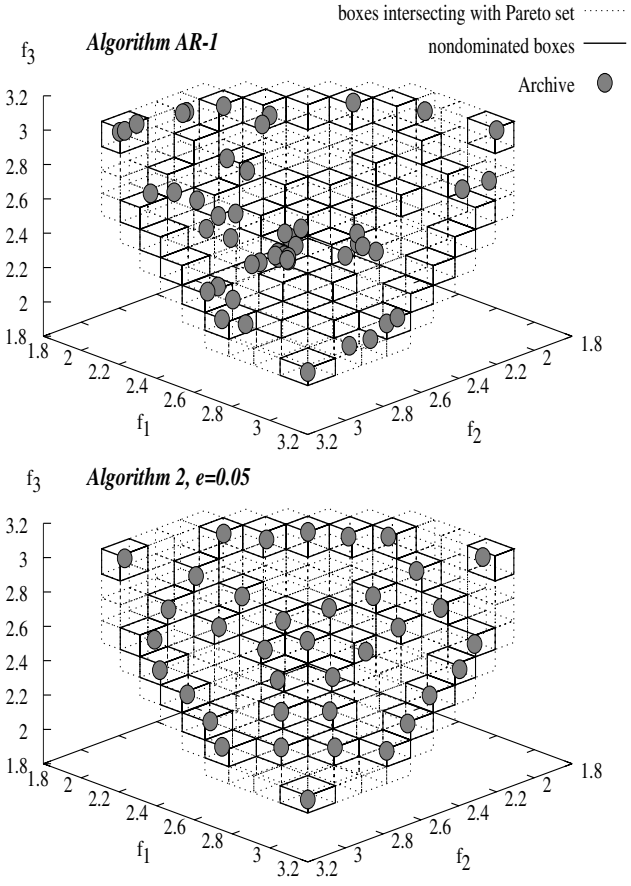


Figure 5: Objective space of MOP (8). The discretization into boxes according to Alg. 3 is indicated by showing all boxes that intersect with the Pareto set F^* in dashed lines. The non-dominated boxes are drawn in bold lines. The circles correspond to the output A of different instances of the iterative search algorithm Alg. 1. For the upper figure an update function according to AR-1 was used, for the lower figure the function according to Alg. 2.

4.2 Distribution Behavior

In order to test for the *distribution* behavior only candidates are taken into account which fulfill the requirements for convergence: Rudolph's and Agapie's algorithm AR-I (Rudolph and Agapie 2000) and Alg. 2. As a test case the following continuous three-dimensional three-objective problem is used:

$$\left.\begin{array}{ll} \text{Max} & f_1(x) = 3 - (1+x_3)\cos(x_1\pi/2)\cos(x_2\pi/2), \\ \text{Max} & f_2(x) = 3 - (1+x_3)\cos(x_1\pi/2)\sin(x_2\pi/2), \\ \text{Max} & f_3(x) = 3 - (1+x_3)\cos(x_1\pi/2)\sin(x_1\pi/2), \\ & 0 \le x_i \le 1, \quad \text{for } i = 1,2,3, \end{array}\right\} \tag{8}$$

The Pareto set of this problem is a surface, a quadrant of the hyper-sphere of radius 1 around $(3,3,3)$. For the results shown in Figure 5 the real-coded NSGA without fitness sharing, crossover with SBX (distribution index $\eta = 5$) and population size 100 was used to generate the candidate solutions. The distribution quality is judged in terms of the ϵ-dominance concept, therefore a discretization of the objective space into boxes (using Alg. 3 with $\epsilon = 0.05$) is plotted instead of the actual Pareto set. As the multiplicative ϵ is used, it can be seen that the box sizes vary and reflect the relative deviations from different parts of the Pareto set. From all boxes intersecting with the Pareto set the non-dominated ones are highlighted. For an ϵ-approximate Pareto set it is now sufficient to have exactly one solution in each of those non-dominated boxes. This condition is fulfilled by the algorithm using the update strategy Alg. 2, leading to an almost symmetric distribution covering all regions. The strategy from AR-1, which does not discriminate among non-dominated points, is sensitive to the sequence of the generated solution and fails to provide an ϵ-approximation of the Pareto set of similar quality even with an allowed archive size of 50.

Looking at the graphs of Algorithm 2, one might have the impression that not all regions of the Pareto set are equally represented by archive members. However, these examples represent *optimal* approximations according to the concepts explained in section 3.2. They are not intended to give a uniform distribution on a (hypothetical) surface that might even not exist as in the discrete case.

4.3 Results

The simulation results support the claims of the preceding sections. The archive updating strategy plays a crucial role for the convergence and distribution properties. The key results are:

- Rudolph's and Agapie's algorithm guarantees convergence, but has no control over the distribution of points.
- The current MOEAs designed for maintaining a good distribution do not fulfill the convergence criterion, as has been demonstrated for SPEA and NSGA-II for a simple test case.
- The algorithm proposed in this paper fulfills both the convergence criterion and the desired distribution control as it always maintains an ϵ-Pareto set of the generated solutions.

5 Possible Extensions

The above baseline algorithms can be extended in several interesting and useful ways. In the following we discuss two examples.

5.1 Steering Search by Defining Ranges of Non-acceptance

In most multi-objective optimization problems, a decision-maker plays an important role. If the complete search space is not of importance to a decision-maker, the above algorithm can be used to search along preferred regions. The concept of ϵ-dominance will then allow pre-specified precisions to exist among the preferred Pareto-optimal vectors.

5.2 Fixed Archive Size by Dynamic Adaptation of ϵ

Instead of predetermining an approximation accuracy ϵ in advance, one might ask whether the algorithm would be able to dynamically adjust its accuracy to always maintain a set of vectors of a given size. A concept like this is implemented in PAES, where the hyper-grid dividing the objective space is adapted to the current ranges given by the non-dominated vectors. However, PAES does not guarantee convergence.

The idea is to start with a minimal ϵ, which is systematically increased every time the number of archived vectors exceeds a predetermined maximum. In Algorithm 2, a simple modification would be to start with a minimal ϵ using a first pair of mutually non-dominated vectors. Afterwards, the increase of ϵ is taken care of by joining boxes and discarding all but the oldest element from the new box.

The joining of boxes could be done in several ways, however for ensuring the convergence property it is important not to move or translate any of the box boundaries, in other words, the assignment of the elements to the boxes must stay the same. A simple implementation could join every second box, while it suffices to join only in the dimensions where the ranges have been exceeded by the new element. This will mean that in the worst case an area will be ϵ-dominated which is almost twice the size of the actual Pareto set in each dimension. A more sophisticated approach would join only two boxes at a time, which would eliminate the over-covering, but involve a complicated book-keeping of several different ϵ values in each dimension.

6 Conclusions

In this study we have addressed the problem of simultaneously achieving convergence and distribution quality when approximating Pareto sets of multi-objective optimization problems. It was shown that none of the existing multi-objective evolutionary algorithms is able to accomplish both tasks.

We proposed the ϵ-(approximate) Pareto set as a solution concept for evolutionary multi-objective optimization that

- is theoretically attractive as it helps to construct algorithms with the desired convergence and distribution properties, and
- is practically important as it works with a solution set with bounded size and predefined resolution.

We constructed the first archive updating strategy that

- can be used in any iterative search process and
- allows for the desired convergence properties while at the same time
- guaranteeing an optimal distribution of solutions.

As we have exclusively dealt with the update operator (or the archiving/selection scheme of the corresponding search and optimization algorithms) so far, all statements had to be done with respect to the generated solutions only. In order to make statements about the convergence to the Pareto set of the whole search space one has of course to include the generator into the analysis. However, with appropriate assumptions (non-vanishing probability measure for the generation of all search points at any time step) it is clear that the probability of not creating a specific point goes to zero as t goes to infinity. Analogously to (Hanne 1999) or (Rudolph and Agapie 2000), results on the limit behavior such as almost sure convergence and stochastic convergence to an ϵ-Pareto set (including all nice features as described in this paper) can be derived.

Though the limit behavior might be of mainly theoretical interest, it is of high practical relevance that now the problem of partial deterioration, which was imminent even in the elitist MOEAs, could be solved. Using the proposed archiving strategy maintaining an ϵ-Pareto set the user can be sure to have in addition to a representative, well distributed approximation also a true elitist algorithm in the sense that no better solution had been found and subsequently lost during the run.

Interesting behaviors occur when there are interactions between the archive and the generator. Allowing the archive members to take part in the generating process has empirically been investigated e.g. by Laumanns et al. (2000, 2001) using a more general model and a parameter called *elitism intensity*. Now, also the theoretical foundation is given so that the archived members are really guaranteed to be the best solutions found.

Acknowledgments

The work has been supported by the Swiss National Science Foundation (SNF) under the ArOMA project 2100-057156.99/1.

References

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: Wiley.

Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI (PPSN-VI)*, pp. 849–858.

Erlebach, T., H. Kellerer, and U. Pferschy (2001). Approximating multi-objective knapsack problems. In *Proceedings of the Seventh International Workshop on Algorithms and Data Structures (WADS 2001)*, Lecture Notes in Computer Science Vol. 2125, Berlin. Springer.

Evtushenko, Y. G. and M. A. Potapov (1987). Methods of numerical solution of multicriterion problem. *Soviet mathematics – doklady 34*, 420–423.

Fonseca, C. M. and P. J. Fleming (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 416–423.

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

Hanne, T. (1999). On the convergence of multiobjective evolutionary algorithms. *European Journal Of Operational Research 117*(3), 553–564.

Helbig, S. and D. Pateva (1994). On several concepts for ϵ-efficiency. *OR Spektrum 16*(3), 179–186.

Horn, J., N. Nafploitis, and D. Goldberg (1994). A niched Pareto genetic algorithm for multi-objective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 82–87.

Knowles, J. D. (2002). *Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization*. Ph. D. thesis, The University of Reading, Deparment of Computer Science.

Knowles, J. D. and D. W. Corne (2000). Approximating the non-dominated front using the Pareto archived evolution strategy. *Evolutionary Computation Journal 8*(2), 149–172.

Laumanns, M., E. Zitzler, and L. Thiele (2000). A unified model for multi-objective evolutionary algorithms with elitism. In *Congress on Evolutionary Computation (CEC 2000)*, Volume 1, Piscataway, NJ, pp. 46–53. IEEE Press.

Laumanns, M., E. Zitzler, and L. Thiele (2001). On the effects of archiving, elitism, and density based selection in evolutionary multi-objective optimization. In E. Zitzler et al. (Eds.), *Evolutionary Multi-Criterion Optimization (EMO 2001), Proc.*, Lecture Notes in Computer Science Vol. 1993, Berlin, pp. 181–196. Springer.

Papadimitriou, C. H. and M. Yannakakis (2000). The complexity of tradeoffs, and optimal access of web sources. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science FOCS 2000*, pp. 86–92.

Reuter, H. (1990). An approximation method for the efficiency set of multiobjective programming problems. *Optimization 21*, 905–911.

Rudolph, G. (1998). On a multi-objective evolutionary algorithm and its convergence to the pareto set. In *IEEE Int'l Conf. on Evolutionary Computation (ICEC'98)*, Piscataway, pp. 511–516. IEEE Press.

Rudolph, G. (2001). Evolutionary search under partially ordered fitness sets. In *Proceedings of the International Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ISI 2001)*, pp. 818–822.

Rudolph, G. and A. Agapie (2000). Convergence properties of some multi-objective evolutionary algorithms. In A. Zalzala and R. Eberhart (Eds.), *Congress on Evolutionary Computation (CEC 2000)*, Volume 2, Piscataway, NJ, pp. 1010–1016. IEEE Press.

Ruhe, G. and B. Fruhwirt (1990). ϵ-optimality for bicriteria programs and its application to minimum cost flows. *Computing 44*, 21–34.

Schaffer, J. D. (1984). *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. Ph. D. thesis, Nashville, TN: Vanderbilt University.

Srinivas, N. and K. Deb (1994). Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation Journal 2*(3), 221–248.

Zitzler, E., K. Deb, L. Thiele, C. A. C. Coello, and D. Corne (Eds.) (2001, March). *Evolutionary Multi-Criterion Optimization (EMO 2001)*, Lecture Notes in Computer Science Vol. 1993, Berlin, Germany. Springer.

Zitzler, E. and L. Thiele (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 3*(4), 257–271.

Maintaining Population Diversity by Minimizing Mutual Information

Yong Liu
The University of Aizu
Tsuruga, Ikki-machi
Aizu-Wakamatsu, Fukushima 965-8580, Japan
yliu@u-aizu.ac.jp

Xin Yao
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham, U.K.
X.Yao@cs.bham.ac.uk

Abstract

Based on negative correlation learning [1] and evolutionary learning, evolutionary ensembles with negative correlation learning (EENCL) was proposed for learning and designing of neural network ensembles [2]. The idea of EENCL is to regard the population of neural networks as an ensemble, and the evolutionary process as the design of neural network ensembles. EENCL used a fitness sharing based on the covering set. Such fitness sharing did not make accurate measurement on the similarity in the population. In this paper, a fitness sharing scheme based on mutual information is introduced in EENCL to evolve a diverse and cooperative population. The effectiveness of such evolutionary learning approach was tested on two real-world problems.

1 Introduction

Neural network ensembles adopt the divide-and-conquer strategy. Instead of using a single network to solve a task, an neural network ensemble combines a set of neural networks which learn to subdivide the task and thereby solve it more efficiently and elegantly [1]. However, designing neural network ensembles is a very difficult task. It relies heavily on human experts and prior knowledge about the problem. Based on negative correlation learning [3, 1] and evolutionary learning, evolutionary ensembles with negative correlation learning (EENCL) was proposed for learning and designing of neural network ensembles [2]. The idea of EENCL is to regard the population of neural networks as an ensemble, and the evolutionary process as the design of neural network ensembles.

The negative correlation learning and fitness sharing [4, 5] were adopted in EENCL to encourage the formation of species in the population. The idea of negative correlation learning is to encourage different individual networks in the ensemble to learn different parts or aspects of the training data, so that the ensemble can better learn the entire training data. In negative correlation learning, the individual networks are trained simultaneously rather than independently or sequentially. This provides an opportunity for the individual networks to interact with each other and to specialize.

Fitness sharing refers to a class of speciation techniques in evolutionary computation. The fitness sharing used in EENCL was based on the idea of the covering set that consists of the same training patterns correctly classified by the shared individuals. This fitness sharing cannot accurately measure the similarity between two individuals. For example, even two individuals have the same covering set, the outputs of two individuals can be quite different. A more accurate similarity measurement between two neural networks in a population can be defined by the explicit mutual information of output variables extracted by two neural networks. The mutual information between two variables, output F_i of network i and output F_j of network j, is given by

$$I(F_i; F_j) = h(F_i) + h(F_j) - h(F_i, F_j) \qquad (1)$$

where $h(F_i)$ is the differential entropy of F_i, $h(F_j)$ is the differential entropy of F_j, and $h(F_i, F_j)$ is the joint differential entropy of F_i and F_j. The equation shows that joint differential entropy can only have high entropy if the mutual information between two variables is low, while each variable has high individual entropy. That is, the lower mutual information two variables have, the more different they are. By minimizing the mutual information between variables extracted by two neural networks, two neural networks are forced to convey different information about some

features of their input.

This paper presents further results on how to evolve a cooperative population of neural networks by minimizing mutual information [6]. Negative correlation learning is firstly analyzed in terms of minimization of mutual information on a regression task. Secondly, a fitness sharing based on mutual information is introduced into EENCL. Through minimization of mutual information, a diverse and cooperative population of neural networks can be evolved by EENCL. The effectiveness of such evolutionary learning approach was tested on two real-world problems.

The rest of this paper is organized as follows: Section 2 explores the connections between the mutual information and the correlation coefficient, and explains how negative correlation learning can be used to minimize mutual information. Section 3 analyzes negative correlation learning via the metrics of mutual information on a regression task. Section 4 describes EENCL for evolving a population of neural networks, and explores the connections between fitness sharing and mutual information. Section 5 presents experimental results on EENCL by minimizing mutual information. Finally, Section 6 concludes with a summary of the paper.

2 Minimizing Mutual Information by Negative Correlation Learning

2.1 Minimization of Mutual Information

Suppose the output F_i of network i and the output F_j of network j are Gaussian random variables. Their variances are σ_i^2 and σ_j^2, respectively. The mutual information between F_i and F_j can be defined by Eq.(1) [7]. The differential entropy $h(F_i)$ and $h(F_j)$ are given by

$$h(F_i) = \frac{1}{2}[1 + \log(2\pi\sigma_i^2)] \tag{2}$$

and

$$h(F_j) = \frac{1}{2}[1 + \log(2\pi\sigma_j^2)] \tag{3}$$

The joint differential entropy $h(F_i, F_j)$ is given by

$$h(F_i, F_j) = 1 + \log(2\pi) + \frac{1}{2}\log|det(\Sigma)| \tag{4}$$

where Σ is the 2-by-2 covariance matrix of F_i and F_j. The determinant of Σ is

$$det(\Sigma) = \sigma_i^2\sigma_i^2(1 - \rho_{ij}^2) \tag{5}$$

where ρ_{ij} is the correlation coefficient of F_i and F_j

$$\rho_{ij} = \frac{E[(F_i - E[F_i])(F_j - E[F_j])]}{\sigma_i^2\sigma_j^2} \tag{6}$$

where E indicates the expectation value. Using the formula of Eq.(5), we get

$$h(F_i, F_j) = 1 + \log(2\pi) + \frac{1}{2}\log[\sigma_i^2\sigma_i^2(1 - \rho_{ij}^2)] \tag{7}$$

By substituting Eqs.(2),(3), and (7) in (1), we get

$$I(F_i; F_j) = -\frac{1}{2}\log(1 - \rho_{ij}^2) \tag{8}$$

From Eq.(8), we may make the following statements:

1. If F_i and F_j are uncorrelated, the correlation coefficient ρ_{ij} is reduced to zero, and the mutual information $I(F_i; F_j)$ becomes very small.
2. If F_i and F_j are highly positively correlated, the correlation coefficient ρ_{ij} is close to 1, and mutual information $I(F_i; F_j)$ becomes very large.

Both theoretical and experimental results [8] have indicated that when individual networks in an ensemble are unbiased, average procedures are most effective in combining them when errors in the individual networks are negatively correlated and moderately effective when the errors are uncorrelated. There is little to be gained from average procedures when the errors are positively correlated. In order to create a population of neural networks that are as uncorrelated as possible, the mutual information between each individual neural network and the rest of population should be minimized. Minimizing the mutual information between each individual neural network and the rest of population is equivalent to minimizing the correlation coefficient between them.

2.2 Negative Correlation Learning

We consider estimating y by forming an neural network ensemble whose output is a simple averaging of outputs F_i of a set of neural networks by given the training data set $D = \{(\mathbf{x}(1), y(1)), \cdots, (\mathbf{x}(N), y(N))\}$. All the individual networks in the ensemble are trained on the same training data set D

$$F(n) = \frac{1}{M}\Sigma_{i=1}^{M} F_i(n) \tag{9}$$

where $F_i(n)$ is the output of individual network i on the nth training pattern $\mathbf{x}(n)$, $F(n)$ is the output of the neural network ensemble on the nth training pattern, and M is the number of individual networks in the neural network ensemble.

The idea of negative correlation learning is to introduce a correlation penalty term into the error function

of each individual network so that the mutual information among the ensemble can be minimized. The error function E_i for individual i on the training data set $D = \{(\mathbf{x}(1), y(1)), \cdots, (\mathbf{x}(N), y(N))\}$ in negative correlation learning is defined by

$$\begin{aligned} E_i &= \frac{1}{N}\Sigma_{n=1}^{N} E_i(n) \\ &= \frac{1}{N}\Sigma_{n=1}^{N}\left[\frac{1}{2}(F_i(n) - y(n))^2 + \lambda p_i(n)\right] \end{aligned} \quad (10)$$

where N is the number of training patterns, $E_i(n)$ is the value of the error function of network i at presentation of the nth training pattern, and $y(n)$ is the desired output of the nth training pattern. The first term in the right side of Eq.(10) is the mean-squared error of individual network i. The second term p_i is a correlation penalty function. The purpose of minimizing p_i is to negatively correlate each individual's error with errors for the rest of the ensemble. The parameter λ is used to adjust the strength of the penalty.

The penalty function p_i has the form

$$p_i(n) = -\frac{1}{2}(F_i(n) - F(n))^2 \quad (11)$$

The partial derivative of E_i with respect to the output of individual i on the nth training pattern is

$$\begin{aligned} \frac{\partial E_i(n)}{\partial F_i(n)} &= F_i(n) - y(n) - \lambda(F_i(n) - F(n)) \\ &= (1-\lambda)(F_i(n) - y(n)) \\ &\quad +\lambda(F(n) - y(n)) \end{aligned} \quad (12)$$

where we have made use of the assumption that the output of ensemble $F(n)$ has constant value with respect to $F_i(n)$. The value of parameter λ lies inside the range $0 \le \lambda \le 1$ so that both $(1-\lambda)$ and λ have nonnegative values. The standard back-propagation (BP) [9] algorithm has been used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual networks is performed simultaneously using Eq.(12) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is called an *epoch*. Negative correlation learning from Eq.(12) is a simple extension to the standard BP algorithm. In fact, the only modification that is needed is to calculate an extra term of the form $\lambda(F_i(n) - F(n))$ for the ith neural network.

From Eqs.(10), (11), and (12), we may make the following observations:

1. During the training process, all the individual networks interact with each other through their penalty terms in the error functions. Each network F_i minimizes not only the difference between $F_i(n)$ and $y(n)$, but also the difference between $F(n)$ and $y(n)$. That is, negative correlation learning considers errors what all other neural networks have learned while training an neural network.

2. For $\lambda = 0.0$, there are no correlation penalty terms in the error functions of the individual networks, and the individual networks are just trained independently using BP. That is, independent training using BP for the individual networks is a special case of negative correlation learning.

3. For $\lambda = 1$, from Eq.(12) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - y(n) \quad (13)$$

Note that the error of the ensemble for the nth training pattern is defined by

$$E_{ensemble} = \frac{1}{2}(\frac{1}{M}\Sigma_{i=1}^{M} F_i(n) - y(n))^2 \quad (14)$$

The partial derivative of $E_{ensemble}$ with respect to F_i on the nth training pattern is

$$\begin{aligned} \frac{\partial E_{ensemble}}{\partial F_i(n)} &= \frac{1}{M}(\frac{1}{M}\Sigma_{i=1}^{M} F_i(n) - y(n)) \\ &= \frac{1}{M}(F(n) - y(n)) \end{aligned} \quad (15)$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ensemble}}{\partial F_i(n)} \quad (16)$$

The minimization of the error function of the ensemble is achieved by minimizing the error functions of the individual networks. From this point of view, negative correlation learning provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual networks.

3 Simulation Results

In order to understand how negative correlation learning minimizes mutual information, this section analyses it through measuring mutual information on a regression task in three cases: noise free condition, small noise condition, and large noise condition.

3.1 Simulation Setup

The regression function investigated here is

$$f(\mathbf{x}) = \frac{1}{13}\left[10sin(\pi x_1 x_2) + 20\left(x_3 - \frac{1}{2}\right)^2 + 10x_4 + 5x_5\right] - 1 \quad (17)$$

where $\mathbf{x} = [x_1, \ldots, x_5]$ is an input vector whose components lie between zero and one. The value of $f(\mathbf{x})$ lies in the interval $[-1, 1]$. This regression task has been used by Jacobs [10] to estimate the bias of mixture-of-experts architectures and the variance and covariance of experts' weighted outputs.

Twenty-five training sets, $(\mathbf{x}^{(k)}(l), y^{(k)}(l))$, $l = 1, \cdots, L$, $L = 500$, $k = 1, \cdots, K$, $K = 25$, were created at random. Each set consisted of 500 input-output patterns in which the components of the input vectors were independently sampled from a uniform distribution over the interval (0,1). In the noise free condition, the target outputs were not corrupted by noise; in the small noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2 = 0.1$ to the function $f(\mathbf{x})$; in the large noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2 = 0.2$ to the function $f(\mathbf{x})$.

A testing set of 1024 input-output patterns, $(\mathbf{t}(n), d(n))$, $n = 1, \cdots, N$, $N = 1024$, was also generated. For this set, the components of the input vectors were independently sampled from a uniform distribution over the interval (0,1), and the target outputs were not corrupted by noise in all three conditions.

Each individual network in the ensemble is a multilayer perceptron with one hidden layer. All the individual networks have five hidden nodes in an ensemble architecture. The hidden node function is defined by the logistic function

$$\varphi(y) = \frac{1}{1 + \exp(-y)} \quad (18)$$

The network output is a linear combination of the outputs of the hidden nodes.

For each estimation of mutual information among an ensemble, twenty-five simulations were conducted. In each simulation, the ensemble was trained on a different training set from the same initial weights distributed inside a small range so that different simulations of an ensemble yielded different performances solely due to the use of different training sets. Such simulation setup follows the suggestions from Jacobs [10].

3.2 Measurement of Mutual Information

The average outputs of the ensemble and the individual network i on the nth pattern in the testing set, $(\mathbf{t}(n), d(n))$, $n = 1, \cdots, N$, are denoted respectively by $\overline{F}(\mathbf{t}(n))$ and $\overline{F}_i(\mathbf{t}(n))$, which are given by

$$\overline{F}(\mathbf{t}(n)) = \frac{1}{K}\Sigma_{k=1}^{K} F^{(k)}(\mathbf{t}(n)) \quad (19)$$

and

$$\overline{F}_i(\mathbf{t}(n)) = \frac{1}{K}\Sigma_{k=1}^{K} F_i^{(k)}(\mathbf{t}(n)) \quad (20)$$

where $F^{(k)}(\mathbf{t}(n))$ and $F_i^{(k)}(\mathbf{t}(n))$ are the outputs of the ensemble and the individual network i on the nth pattern in the testing set from the kth simulation, respectively, and $K = 25$ is the number of simulations. The correlation coefficient between network i and network j is given by

$$\rho_{ij} = \frac{\Sigma_{n=1}^{N}\Sigma_{k=1}^{K}\left(F_i^{(k)}(\mathbf{t}(n)) - \overline{F}_i(\mathbf{t}(n))\right)}{\sqrt{\Sigma_{n=1}^{N}\Sigma_{k=1}^{K}\left(F_i^{(k)}(\mathbf{t}(n)) - \overline{F}_i(\mathbf{t}(n))\right)^2}} \times \frac{\left(F_j^{(k)}(\mathbf{t}(n)) - \overline{F}_j(\mathbf{t}(n))\right)}{\sqrt{\Sigma_{n=1}^{N}\Sigma_{k=1}^{K}\left(F_j^{(k)}(\mathbf{t}(n)) - \overline{F}_j(\mathbf{t}(n))\right)^2}} \quad (21)$$

From Eq.(6), the integrated mutual information among the ensembles can be defined by

$$E_{mi} = -\frac{1}{2}\Sigma_{i=1}^{M}\Sigma_{j=1, j\neq i}^{M} log(1 - \rho_{ij}^2) \quad (22)$$

We may also define the integrated mean-squared error (MSE) on the testing set as

$$E_{test\_mse} = \frac{1}{N}\Sigma_{n=1}^{N}\frac{1}{K}\Sigma_{k=1}^{K}\left(F^{(k)}(\mathbf{t}(n)) - d(n)\right)^2 \quad (23)$$

The integrated mean-squared error E_{train} on the training set is given by

$$E_{train\_mse} = \frac{1}{L}\Sigma_{l=1}^{L}\frac{1}{K}\Sigma_{k=1}^{K}\left(F^{(k)}(\mathbf{x}^{(k)}(l)) - y^{(k)}(l)\right)^2 \quad (24)$$

3.3 Results in the Noise Free Condition

The results of negative correlation learning in the noise free condition for the different values of λ at epoch 2000 are given in Table 1. The results suggest that

Table 1: The Results of Negative Correlation Learning in the Noise Free Condition for Different λ Values at Epoch 2000.

λ	E_{mi}	E_{test_mse}	E_{train_mse}
0	0.3706	0.0016	0.0013
0.25	0.1478	0.0013	0.0010
0.5	0.1038	0.0011	0.0008
0.75	0.1704	0.0007	0.0005
1	0.6308	0.0002	0.0001

Table 2: The Results of Negative Correlation Learning in the Small Noise Condition for Different λ Values at Epoch 2000.

λ	E_{mi}	E_{test_mse}	E_{train_mse}
0	6.5495	0.0137	0.0962
0.25	3.8761	0.0128	0.0940
0.5	1.4547	0.0124	0.0915
0.75	0.3877	0.0126	0.0873
1	0.2431	0.0290	0.0778

both E_{train_mse} and E_{test_mse} appeared to decrease with increasing value of λ. The mutual information E_{mi} among the ensemble decreased as the value of λ increased when $0 \leq \lambda \leq 0.5$. However, when λ increased further to 0.75 and 1, the mutual information E_{mi} had larger values. The reason of having larger mutual information at $\lambda = 0.75$ and $\lambda = 1$ is that some correlation coefficients had negative values and the mutual information depends on the absolute values of correlation coefficients.

In order to find out why E_{train_mse} decreased with increasing value of λ, the concept of capability of a trained ensemble is introduced. The capability of a trained ensemble is measured by its ability of producing correct input-output mapping on the training set used, specifically, by its integrated mean-squared error E_{train_mse} on the training set. The smaller E_{train_mse} is, the larger capability the trained ensemble has.

3.3.1 Results in the Noise Conditions

Table 2 and Table 3 compare the performance of negative correlation learning for different strength parameters in both small noise (variance $\sigma^2 = 0.1$) and large noise (variance $\sigma^2 = 0.2$) conditions. The results show that there were same trends for E_{mi}, E_{test_mse}, and E_{train_mse} in both noise free and noise conditions when $\lambda \leq 0.5$. That is, E_{mi}, E_{test_mse}, and E_{train_mse} appeared to decrease with increasing value of λ. However, E_{test_mse} appeared to decrease first and then increase with increasing value of λ.

Table 3: The Results of Negative Correlation Learning in the Large Noise Condition for Different λ Values at Epoch 2000.

λ	E_{mi}	E_{test_mse}	E_{train_mse}
0	6.7503	0.0249	0.1895
0.25	3.9652	0.0235	0.1863
0.5	1.6957	0.0228	0.1813
0.75	0.4341	0.0248	0.1721
1	0.2030	0.0633	0.1512

In order to find out why E_{test_mse} showed different trends in noise free and noise conditions when $\lambda = 0.75$ and $\lambda = 1$, the integrated mean-squared error E_{train_mse} on the training set was also shown in Tables 1, 2, and 3. When $\lambda = 0$, the neural network ensemble trained had relatively large E_{train_mse}. It indicated that the capability of the neural network ensemble trained was not big enough to produce correct input-output mapping (i.e., it was underfitting) for this regression task. When $\lambda = 1$, the neural network ensemble trained learned too many specific input-output relations (i.e., it was overfitting), and it might memorize the training data and therefore be less able to generalize between similar input-output patterns. Although the overfitting was not observed for the neural network ensemble used in noise free condition, too large capability of the neural network ensemble will lead to overfitting for both noise free and noise conditions because of the ill-posedness of any finite training set [11].

Choosing a proper value of λ is important, and also problem dependent. For the noise conditions used for this regression task and the ensemble architectured used, the performance of the ensemble was optimal for $\lambda = 0.5$ among the tested values of λ in the sense of minimizing the MSE on the testing set.

4 Evolving Neural Network Ensembles

In EENCL [2], an evolutionary algorithm based on evolutionary programming [12] has been used to search for a population of diverse individual neural networks that solve a problem together. Two major issues were addressed in EENCL, including exploitation of the interaction between individual neural design and combination, and automatic determination of the number of individual neural networks in an ensemble. The major steps of EENCL are given as follows [4]:

1. Generate an initial population of M neural networks, and set $k = 1$. The number of hidden nodes for each neural network, n_h, is specified by the user. The random initial weights are distributed uniformly inside a small range.

2. Train each neural network in the initial population on the training set for a certain number of epochs using negative correlation learning. The number of epochs, n_e, is specified by the user.

3. Randomly choose a group of n_b neural networks as parents to create n_b offspring neural networks by Gaussian mutation.

4. Add the n_b offspring neural networks to the population and train the offspring neural networks using negative correlation learning while the remaining neural networks' weights are frozen.

5. Calculate the fitness of $M + n_b$ neural networks in the population and prune the population to the M fittest neural networks.

6. Go to the next step if the maximum number of generations has been reached. Otherwise, $k = k + 1$ and go to Step 3.

7. Form species using the k-means algorithm.

8. Combining species to form the ensembles.

There are two levels of adaptation in EENCL: negative correlation learning at the individual level and evolutionary learning based on evolutionary Forming species by using the k-means algorithm in EENCL [2] is not considered in this paper.

Fitness sharing used in EENCL is based on the idea of covering the same training patterns by shared individuals. The procedure of calculating shared fitness is carried out pattern-by-pattern over the training set. If one training pattern is learned correctly by p individuals in the population, each of these p individuals receives fitness $1/p$, and the rest of the individuals in the population receive zero fitness. Otherwise, all the individuals in the population receive zero fitness. The fitness is summed over all training patterns.

Rather than using the fitness sharing based on the covering set, a fitness sharing based on the minimization of mutual information was introduced in EENCL [6]. In order to create a population of neural networks that are as uncorrelated as possible, the mutual information between each individual neural network and the rest of population should be minimized. The fitness f_i of individual network i in the population can therefore be evaluated by the mutual information:

$$f_i = \frac{1}{\sum_{j \neq i} I(F_i, F_j)} \tag{25}$$

Minimization of mutual information has the similar motivations as fitness sharing. Both of them try to generate individuals that are different from others, though overlaps are allowed.

5 Experimental Studies

This section investigates EENCL with minimization of mutual information on two benchmark problems: the Australian credit card assessment problem and the diabetes problem. Both data sets were obtained from the UCI machine learning benchmark repository. They are available by anonymous ftp at ics.uci.edu (128.195.1.1) in directory /pub/machine-learning-databases.

The Australian credit card assessment problem is to assess applications for credit cards based on a number of attributes. There are 690 patterns in total. The output has two classes. The 14 attributes include 6 numeric values and 8 discrete ones, the latter having from 2 to 14 possible values.

The diabetes data set is a two-class problem that has 500 examples of class 1 and 268 of class 2. There are 8 attributes for each example. The data set is rather difficult to classify. The so-called "class" value is really a binarized form of another attribute that is itself highly indicative of certain types of diabetes but does not have a one-to-one correspondence with the medical condition of being diabetic.

In order to tell the difference between EENCL and EENCL with minimization of mutual information. We name the later approach as EENCLMI. The experimental setup is the same as the previous experimental setup described in [13, 2]. The n-fold cross-validation technique [14] was used to divide the data randomly into n mutually exclusive data groups of equal size. In each train-and-test process, one data group is selected as the testing set, and the other $(n-1)$ groups become the training set. The estimated error rate is the average error rate from these n groups. In this way, the error rate is estimated efficiently and in an unbiased way. The parameter n was set to be 10 for the Australian credit card data set, and 12 for the diabetes data set, respectively.

All parameters used in EENCLMI except for the number of training epochs were set to be the same for both problems: the population size M (25), the number of

Table 4: Comparison of Accuracy Rates between EENCLMI and EENCL for the Australian Credit Card Data Set. The Results Are Averaged on 10-Fold Cross-Validation. *Mean* and *SD* Indicate the Mean Value and Standard Deviation, Respectively.

	Simple Averaging		Majority Voting		Winner-Takes-All	
Methods	Mean	SD	Mean	SD	Mean	SD
EENCLMI	0.864	0.038	0.870	0.040	0.868	0.039
EENCL	0.855	0.039	0.857	0.039	0.865	0.028

Table 5: Comparison of Accuracy Rates between EENCLMI and EENCL for the Diabetes Data Set. The Results Are Averaged on 12-Fold Cross-Validation. *Mean* and *SD* Indicate the Mean Value and Standard Deviation, Respectively.

	Simple Averaging		Majority Voting		Winner-Takes-All	
Methods	Mean	SD	Mean	SD	Mean	SD
EENCLMI	0.771	0.049	0.777	0.046	0.773	0.051
EENCL	0.766	0.039	0.764	0.042	0.779	0.045

generations (200), the reproduction block size n_b (2), the strength parameter λ (0.5), the minimum number of cluster sets (3), and the maximum number of cluster sets (25). The number of training epochs n_e was set to 3 for the Australian credit card data set, and 15 for the diabetes data set. The used neural networks in the population are multilayer perceptrons with one hidden layer and five hidden nodes. These parameters were selected after some preliminary experiments. They were not meant to be optimal.

5.1 Experimental Results

Tables 4–5 show the results of EENCLMI for the two data sets, where the ensembles were constructed by the whole population in the last generation. Three combination methods for determining the output of the ensemble have been investigated in EENCLMI. The first is simple averaging. The output of the ensemble is formed by a simple averaging of output of individual neural networks in the ensemble. The second is majority voting. The output of the greatest number of individual neural networks will be the output of the ensemble. If there is a tie, the output of the ensemble is rejected. The third is winner-takes-all. For each pattern of the testing set, the output of the ensemble is only decided by the individual neural network whose output has the highest activation. The *accuracy rate* refers to the percentage of correct classifications produced by EENCLMI. In comparison with the accuracy rates obtained by three combination methods, majority voting and winner-takes-all outperformed simple averaging on both problems. Simple averaging is more suitable to the regression type of tasks. Because both problems studied in this paper are classification tasks, majority voting and winner-takes-all are better choices.

Tables 4–5 compare the results produced EENCLMI and EENCL using three combination methods. Majority voting supports EENCLMI, while winner-takes-all favors EENCL. Since the only difference between EENCLMI and EENCL is the fitness sharing scheme used, the results suggest that combination methods and fitness sharing are closely related to each other. Further studies are needed to probe the relationship of these two.

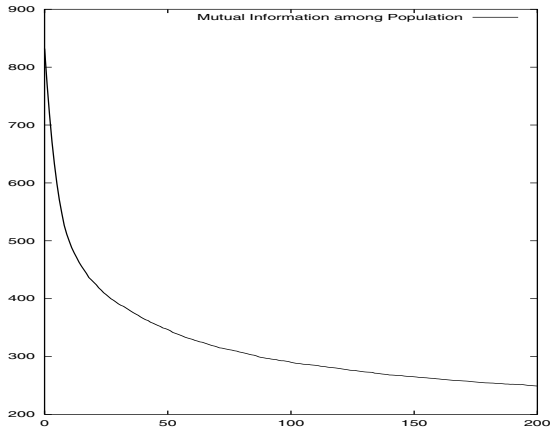


Figure 1: The Evolution of the Mean of Sum of the Mutual Information among the Population for the Australian Credit Card Data Set. The Mean Is Averaged on 10-fold Cross-Validation. The Vertical Axis Is the Mutual Information Value and the Horizontal Axis Is the Number of Generations.

In order to observe the evolutionary process of the mutual information among the population in EENCLMI, Figure 1 show the evolution of the mean of sum of the mutual information among the population for the Australian credit card data set. The sum of the mutual information among the population is calculated by

$$I_{population} = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1, j \neq i}^{M} I(F_i, F_j) \qquad (26)$$

where F_i is the vector formed by the output of network i on the training set, and F_j is the vector formed by the output of network j on the training set. The mean of $I_{population}$ is averaged on 10-fold cross-validation. The evolutionary processes clearly shows that the value of mutual information among the population steadily decreased through the whole evolution.

6 Conclusions

Minimization of mutual information has been introduced as a fitness sharing scheme in EENCL. Compared with the fitness sharing based on the covering set originally used in EENCL [2], mutual information provides more accurate measurement on the similarity. By minimizing mutual information, a diverse population can be evolved.

This paper has also analyzed negative correlation learning in terms of mutual information on a regression task in the different noise conditions. Unlike independent training which creates larger mutual information among the ensemble, negative correlation learning can produce smaller mutual information among the ensemble.

References

[1] Y. Liu and X. Yao. Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):716–725, 1999.

[2] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.

[3] Y. Liu and X. Yao. Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems*, 4:176–185, 1998.

[4] X. Yao, Y. Liu, and P. Darwen. How to make best use of evolutionary learning. In R. Stocker, H. Jelinek, and B. Durnota, editors, *Complex Systems: From Local Interactions to Global Phenomena*, pages 229–242. IOS Press, Amsterdam, 1996.

[5] Y. Liu and X. Yao. Towards designing neural network ensembles by evolution. In *Parallel Problem Solving from Nature — PPSN V: Proc. of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 623–632. Springer-Verlag, Berlin, 1998.

[6] Y. Liu, Q. Zhao X. Yao, and T. Higuchi. Evolving a cooperative population of neural networks by minimizing mutual information. In *Proc. of the 2001 Conference on Evolutionary Computation*, pages 384–389. IEEE Press, 2001.

[7] J. C. A. van der Lubbe. *Information Theory*. Prentice-Hall International, Inc., 2nd edition, 1999.

[8] R. T. Clemen and R. .L Winkler. Limits for the precision and value of information from dependent sources. *Operations Research*, 33:427–442, 1985.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[10] R. A. Jacobs. Bias/variance analyses of mixture-of-experts architectures. *Neural Computation*, 9:369–383, 1997.

[11] J. H. Friedman. An overview of predictive learning and function approximation. In V. Cherkassky, J. H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, pages 1–61. Springer-Verlag, Heidelberg, Germany, 1994.

[12] D. B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. IEEE Press, New York, NY, 1995.

[13] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Limited, London, 1994.

[14] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

Increasing Robustness of Genetic Algorithm

Jiangming Mao, Kotaro Hirasawa, Jinglu Hu and Junichi Murata
Graduate School of Information Science and Electrical Engineering, Kyushu University,
6-10-1, Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
mao@cig.ees.kyushu-u.ac.jp

Abstract

Genetic algorithms are often well suited for optimization problems because of their parallel searching and evolutionary ability. Crossover and mutation are believed to be the main exploration operators in GA. In this paper, we focus on how crossover and mutation work in GA and investigate their effect on bit's frequency of the population. To increase robustness against uncertainty of GA, a new recombination method based on bit's frequency of the population and a new robust generation strategy were proposed. The proposed methods were tested on the problem with many local minima. Simulation results demonstrate the effectiveness of the proposed methods.

1 Introduction

Genetic algorithm (GA) is a random searching method with some special features. One feature is that GAs are versatile evolutionary computation techniques largely based on the principle of survival of the fittest [1]. Another is the genetic operators such as crossover and mutation. When using GA for solving a given problem, the user has to design so many parts to make GA effective, such as the number of population, population size, mutation rate, crossover rate, selection pressure and selection scheme. However, GA can not always get good solutions we want, because it is difficult for users to design an effective GA which is a random searching method.

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[2]. In GA, the crossover and mutation are believed to be the main exploration operators in the working of GA as an optimization tool. In this paper, we focus on the variance of the distribution of the individuals on a hyper-plane, through a new way to investigate how the mutation and crossover work in GA. Furthermore, a new **rec**ombination method based on **b**it's **f**requency (RCBF) was proposed, which can make the population distribute more uniformly than the "conventional" crossover such as one point crossover, two point crossover and uniform crossover. In addition, because all messages from the population are stored in bit's frequency, a new **r**obust **g**eneration **s**trategy (RGS) is proposed where the $t+m$ th generation is determined not only by the $t+m-1$ th generation but also by generations from the t th to the $t+m-2$ th. According to the simulation results, we can find GA by using RCBF and RGS can search for the solutions more robustly than "conventional" GA, especially when the feasible solution space is very large.

This paper is organized as follows. Next section is about a new way of analyzing crossover and mutation of GA. Section 3 introduces RCBF and give some simulation results. Section 4 introduces RGS and give some simulation results. The last section offers concluding remarks and future perspectives.

2 A New Way of Analyzing GA

The GA studied in this paper is the one similar to Simple Genetic Algorithm defined in [2].

2.1 Mathematical description

A k th binary individual X_k in a population can be given by

$$X_k = (x_k^1, \ldots, x_k^j, \ldots, x_k^L), \tag{1}$$

where L is the length of the binary individual, x_k^j stands for the j th bit of the k th individual. A pop-

ulation $\vec{X}$ can be defined as

$$\vec{X} = (X_1, \ldots, X_k, \ldots, X_N), \tag{2}$$

where N is the population size. The feasible space of bit x_k^j is $\{0,1\}$. The feasible space of the individual X_k is $\{0,1\}^L$.

Definition 2.1 (Bit's frequency) Let $f_{\vec{X}}^j$ be the j th bit's frequency of the population $\vec{X}$, where

$$f_{\vec{X}}^j = \frac{1}{N}\sum_{k=0}^{N} x_k^j. \tag{3}$$

The feasible space S_f of $f_{\vec{X}}^j$ is $[0,1]$. The bit's frequency string $F_{\vec{X}}$ can be given by

$$F_{\vec{X}} = (f_{\vec{X}}^1, \ldots, f_{\vec{X}}^j, \ldots, f_{\vec{X}}^L), \tag{4}$$

where the feasible space S_f^L of the bit's frequency string is $[0,1]^L$.

If the population is distributed in Z^L, the population $\vec{X}$ is a set of the vertex of the unit-box with L dimensions. The bit's frequency string $F_{\vec{X}}$ can be represented in R^L. We can see a population is a dynamical structure with a centre of gravity $F_{\vec{X}}$ in R^L. To investigate the effect of mutation and crossover, we will do some research on the variance of the centre of gravity of the population.

2.2 Crossover Operator

The crossover operator T_c is a very complex operator to recombine the gene of each individual in the population. There exist a number of crossover operators in the GA literature, such as one point crossover, two point crossover and uniform crossover. According to the quality of crossover, we know the crossover operators don't change the bit's frequency string. To investigate the effect of the crossover operator, let us see the next definition.

Definition 2.2 If $F_{\vec{X}}$=$F_{\vec{Y}}$, we can say the population $\vec{X}$ is similar to the population $\vec{Y}$, denoted by $\vec{X} \sim \vec{Y}$.

Because we can not derive $\vec{X} = \vec{Y}$ ($\vec{X}$ and $\vec{Y}$ are the same) from $\vec{X} \sim \vec{Y}$, the crossover operators can change the population from one case to another with the same bit's frequency string.

Definition 2.3 (Distribution state function) we use a two-order function to show the distribution state $E_{\vec{Y}}$ of the population $\vec{Y}$ as follows,

$$E_{\vec{Y}} = \frac{N_{(0,\ldots,0)}^2 + N_{(0,\ldots,1)}^2 + \ldots + N_{(1,\ldots,1)}^2}{N^2}, \tag{5}$$

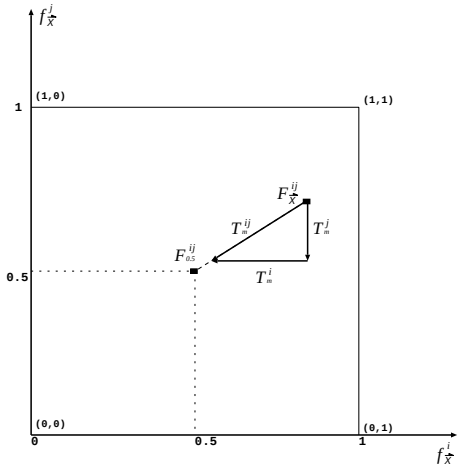


Fig. 1. Mechanism of mutation in $F_{\vec{X}}$.

where $N_{(\ldots)}$ is the overlapping number of the individual $(\ldots)$, and $N_{(0,\ldots,0)} + N_{(0,\ldots,1)} + \ldots + N_{(1,\ldots,1)} = N$.

If the bit's frequency string $F_{\vec{Y}}$ of the population $\vec{Y}$ is determined, using the following $N_{(y^1,\ldots,y^j,\ldots,y^L)}$

$$N_{(y^1,\ldots,y^j,\ldots,y^L)} = \lfloor (\prod_{j=1}^{L} \hat{f}_{\vec{Y}}^j)\ N + 0.5 \rfloor, \tag{6}$$

where

$$\hat{f}_{\vec{Y}}^j = \begin{cases} f_{\vec{Y}}^j & \text{if } y^j = 1 \\ 1 - f_{\vec{Y}}^j & \text{if } y^j = 0 \end{cases}$$

then $E_{\vec{Y}}$ has the minimum value, denoted by $E_{\min}$. Furthermore, as $E_{\min}$ is determined by the bit's frequency string $F_{\vec{Y}}$, when $F_{\vec{Y}} = F_{0.5}$ (where $F_{0.5} = (0.5, \ldots, 0.5)$), $E_{\min}$ will be the most minimum.

Supposing we use T_c to make the population $\vec{X}$ crossover d times where $d = 1, 2, \ldots$, then we can give $E_{\vec{X}} \to E_{\min}$ when $d \to \infty$. In other words, the crossover operator has an ability to make the population distribute uniformly without changing the bit's frequency string.

2.3 Mutation Operator

The mutation operator is a force T_m (a vector quantity) to maintain the diversity in the population and is used with a small probability, p_m. To give the direction and strength of the mutation force, we describe the population $\vec{X}$ onto the plane, for example, $(f_{\vec{X}}^i, f_{\vec{X}}^j)$ plane shown in Fig.1, where $f_{\vec{X}}^i$ and $f_{\vec{X}}^j$ are the lateral and vertical coordinates. In Fig.1, $F_{0.5}^{ij}$ is the two dimensional point of $F_{0.5}$, $F_{\vec{X}}^{ij}$ is the two dimensional

point of $F_{\vec{X}}$, T_m^{ij} is the two dimention vector quantity of T_m on the plane $(f_{\vec{X}}^i, f_{\vec{X}}^j)$, T_m^i and T_m^j are the component quantities of T_m on the $f_{\vec{X}}^i$ and $f_{\vec{X}}^j$ coordinates respectively. We can easily give T_m^i and T_m^j as follows,

$$T_m^i = \frac{N_{(x^i=1)} - N_{(x^i=0)}}{N}\ p_m,$$
$$T_m^j = \frac{N_{(x^j=1)} - N_{(x^j=0)}}{N}\ p_m,$$

where $N_{(*)}$ is the number of individuals of the population $\vec{X}$ where x^i or x^j is equal to 0 or 1, therefore $N_{(x^i=1)} + N_{(x^i=0)} = N_{(x^j=1)} + N_{(x^j=0)} = N$. So we can easily give the strength of T_m^{ij} as follows,

$$\begin{aligned} |T_m^{ij}| &= \sqrt{\left(\frac{N_{(x^i=1)} - N_{(x^i=0)}}{N}\right)^2 + \left(\frac{N_{(x^j=1)} - N_{(x^j=0)}}{N}\right)^2}\, p_m \\ &= 2\ |\overrightarrow{F_{\vec{X}}^{ij} F_{0.5}^{ij}}|\ p_m, \end{aligned}$$

where $\overrightarrow{F_{\vec{X}}^{ij} F_{0.5}^{ij}}$ is a vector from the point $F_{\vec{X}}^{ij}$ to $F_{0.5}^{ij}$, $|\overrightarrow{F_{\vec{X}}^{ij} F_{0.5}^{ij}}|$ is the length of the vector $\overrightarrow{F_{\vec{X}}^{ij} F_{0.5}^{ij}}$. The direction of T_m^{ij} can be easily demonstrated to be the same as the direction of the vector $\overrightarrow{F_{\vec{X}}^{ij} F_{0.5}^{ij}}$. Generally, we can easily give the strength of T_m as follows,

$$|T_m| = 2\ |\overrightarrow{F_{\vec{X}} F_{0.5}}|\ p_m, \tag{7}$$

where $|\overrightarrow{F_{\vec{X}} F_{0.5}}|$ is the distance between point $F_{\vec{X}}$ and $F_{0.5}$. The direction of T_m is from point $F_{\vec{X}}$ to $F_{0.5}$.

According to Eq.(7), we can see mutation operator can change the bit's frequency string, where the strength of the mutation force is changed proportionally along with the convergence status (represented by $|\overrightarrow{F_{\vec{X}} F_{0.5}}|$) of the population and the direction of the mutation force is always from the point $F_{\vec{X}}$ to $F_{0.5}$. In other words, the mutation operator can change $E_{\min}$ which is determined by the bit's frequency string.

Furthermore, the mutation operator has another ability which is the same as the crossover operator. For example, if the bit's frequency string $F_{\vec{X}}$=$F_{0.5}$ and $E_{\vec{X}} > E_{\min}$, mutating the population $\vec{X}$ infinite times, $E_{\vec{X}}$ should be $E_{\min}$ without changing the bit's frequency string. Generally, this ability of the mutation operator exists in the case even when $F_{\vec{X}} \neq F_{0.5}$ and is smaller and smaller along with the concentration of the population.

There exist two kinds of abilities of the mutation operator, so we can separate the mutation operator into two parts: the first part which is determined by $|N_{(x^*=1)} - N_{(x^*=0)}|$ can change the the bit's frequency string while the second part which is determined by $\min\{N_{(x^*=1)}, N_{(x^*=0)}\}$ can make the population distribute uniformly without changing the bit's frequency string.

2.4 Concentration of the population

In a searching process by using GAs, the variance of the individuals' fitness is reduced due to two factors. One factor is selection pressure producing multiple copies of fitter population members while the other factor is independent of population member's fitness and is due to the stochastic nature of the selection operator, –genetic drift.[3]

Under the operation of selection, the fitter member of the population have higher chance of producing more offspring than the less member. If the selection pressure is greater than the mutation and crossover force, selection pressure makes all individuals of the population concentrate to the optimal points. We can separate selection methods into two main categories: using ranking methods [4][5] and not using ranking methods. The selection pressure without ranking methods is determined by the difference of the individuals' fitness, so it changes along with the evolutionary process. The selection pressure with ranking methods doesn't change along with the difference of the individuals' fitness. So the selection pressure with ranking methods can be more easily controlled than without ranking methods. But it takes much time to calculate the rank of each individual. Genetic drift makes the population concentrate randomly. The effect of genetic drift is not shown very clearly when the objective function is a unimodal function. But for multimodal functions, genetic drift should make the population concentrate to one of the optimal solutions randomly.

Anyway, selection pressure and genetic drift make the population concentrate. In other words, they make the $|\overrightarrow{F_{\vec{X}} F_{0.5}}|$ and $E_{\vec{X}}$ large.

3 Recombination Method based on Bit's Frequency

3.1 Species and Sampling

From the previous section, when $\vec{X} \sim \vec{Y}$ and $E_{\vec{Y}} = E_{\min}$, the mutation (the second part) and crossover operators make the population $\vec{X}$ approach $\vec{Y}$. In other words, the population $\vec{Y}$ is stabler than the population $\vec{X}$. Using this, a new recombination method stated in 3.2 is proposed.

Definition 3.1 (Species) A species can be defined as: a group of individuals that 1)actually or potentially

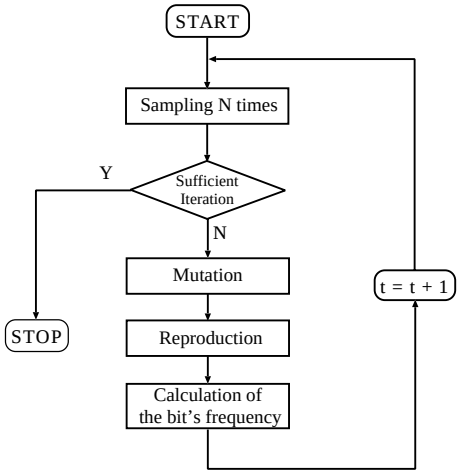


Fig. 2. Structure of the proposed algorithm based on bit's frequency.

interbreed with each other but not with other groups, 2)the distinction from other groups is the bit's frequency string F, where $F = (f^1, \ldots, f^j, \ldots, f^L)$, f^j is the frequency when '1' appears in the j th bit of the species

Definition 3.2 (Sampling) Sampling is an operator such as getting sample individuals from a species. This operator is given as follows: each bit of sample individuals is determined randomly according to the bit's frequency string F of the species.

According to the definitions, a species is a population with a certain bit's frequency string. The individuals of the species can crossover with each other but can not do with other species. Each bit's of the individual X of the species can be determined by this bit's frequency. It means the distribution of the individuals can satisfy Eq.6 and is not changed by crossover and the second part of mutation because $E_{\vec{X}}$ is minimum.

3.2 Flow of the proposed method

An simple genetic algorithm by using RCBF is shown in this subsection. The basic structure is shown in Fig.2, where the initial value of the bit's frequency string F is $F_{0.5}$. One iteration at the t th generation can be described as follows: 1)after sampling N times according to the bit's frequency string $F_{(t)}$ we can get a population $\vec{X}_{(t)}$ with N members; 2)after mutation and reproduction we can get a population $\vec{X'}_{(t)}$; 3)we can calculate the bit's frequency string $F_{\vec{X'}_{(t)}}$ of the population $\vec{X'}_{(t)}$ and set the bit's frequency string for the next generation.

In fact, we can consider RCBF as the strongest

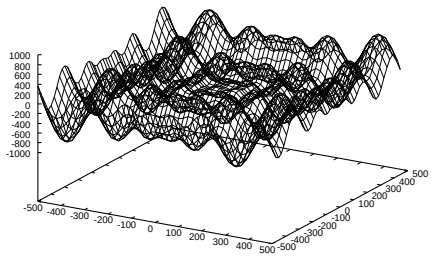


Fig. 3. The two-dimensional version of f_8.

crossover, because it can make the population $\vec{X'}_{(t)}$ to be the population $\vec{X}_{(t+1)}$ which is distributed the most uniformly. It means RCBF can search more points than "conventional" crossover method such as the uniform crossover method. In other words, although the feasible search points of RCBF and uniform crossover are the same, RCBF can search for more of them than the uniform crossover. Especially when the feasible solution space is very large, doing more searching is very useful to increase the searching ability and the robustness of GA. Furthermore, because all messages from the environment are stored in the bit's frequency, sometimes, it is very useful to use RCBF in order to decrease the memory and time required for calculation.

3.3 Experiments

Generalized Schwefel's Problem which was examined in [6]-[7] is used in our experimental studies.

$$\begin{aligned} \min f_8(x) &= -\sum_{i=1}^{K}(x_i \sin(\sqrt{|x_i|})), \\ \text{where} \quad & K = 1, 2, \ldots, 30 \\ & -500 < x_i < 500 \end{aligned}$$

This function is a multimodal function with many local minima, where the number of local minima increases exponentially as the dimension of the function increases like 7^K. The global minimal function's value is $K \times 418.98289$. Fig.3 shows the two-dimensional version of f_8. To analyze the genetic algorithm by using the species concept, we can do some comparisons of the proposed method with the uniform crossover method.

3.3.1 Parameter Values

- Population size: Since the problem dimensions are high, we choose a moderate population size N=200;
- Representation: Each variable has 30 bits, so the length of the individuals is $30 \times K$.

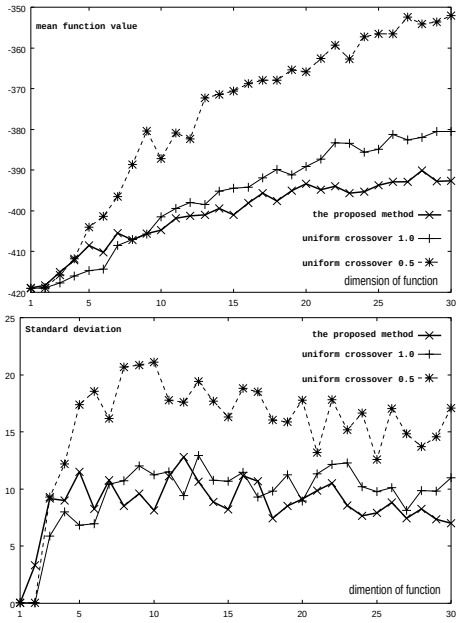


Fig. 4. Comparison between the proposed recombination method and the uniform crossover method in terms of the mean and the standard deviation of function value.

- Crossover rate: We set crossover rate 1.0 and 0.5 for the uniform crossover method respectively.
- Mutation probability: We choose $p_m = \frac{1}{L}$.
- Selection pressure: We use the nonlinear ranking method[5] where the selection probability of the k th individual can be calculated as $p_k = c \times (1-c)^{i-1}$, i is the rank of the k th individual. We set the parameter $c = 0.05$.
- Iteration: The stopping generation is $\lfloor 50 \times \sqrt{K} + 0.5 \rfloor$.

3.3.2 Discussions

We performed 50 independent runs for the proposed method and uniform crossover method from $K = 1$ to $K = 30$ and recorded 1)mean function value (the mean value of the best individual of the last generation over 50 runs) and 2)the standard deviation of function value (the standard deviation of the best individual of the last generation over 50 runs). Fig.4 shows the simulation results. The upper part shows the mean function value $\bar{f}$ where the lateral coordinate is the dimension of the test function, while the lower part shows the standard deviation of the function value σ_f.

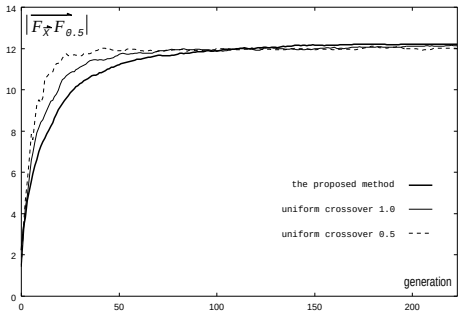


Fig. 5. Comparison of the convergence speed between the proposed recombination method and the uniform crossover method.

They can be calculated as follows:

$$\bar{f} = \frac{1}{50}\sum_{i=1}^{50}\frac{f_i}{K},$$

$$\sigma_f = \sqrt{\frac{1}{50}\sum_{i=1}^{50}(\frac{f_i}{K} - \bar{f})^2}.$$

According to these results, we can see

- when the dimension of the function is large, the mean function value of the proposed method is smaller than that of the uniform method with crossover rate 1.0, followed by that of the uniform crossover with crossover rate 0.5. It means that the search ability of RCBF is strongest compared with the uniform crossover. The standard deviation of RCBF is smaller than that of the uniform method with 1.0 crossover rate, followed by that of the uniform method with 0.5 crossover rate. It means that RCBF can increase the robustness against uncertainty of GA.
- when the dimension of function is small, the mean function value and the standard deviation of the proposed method is larger than those of the uniform crossover method.

Fig.5 shows the simulation results of the convergence speed which was randomly selected when $K = 20$, where the lateral coordinate is generation and the vertical coordinate is $|\overrightarrow{F_{\vec{X}}F_{0.5}}|$. According to these results, we can see that the population concentrating rates of the proposed method is slower than that of the uniform crossover with 1.0 crossover rate, followed by that of the uniform crossover with 0.5 crossover rate.

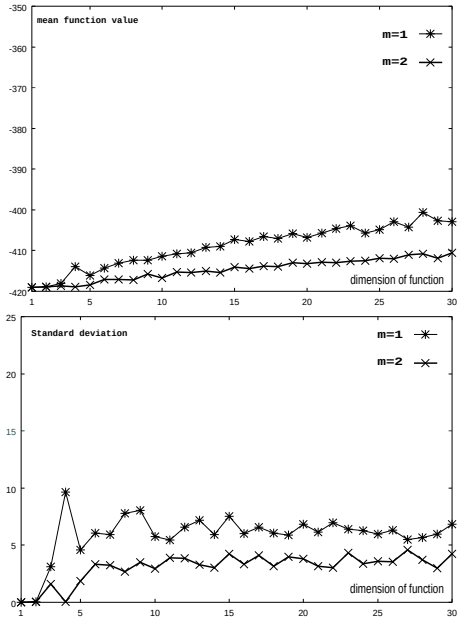


Fig. 6. Mean and standard deviation by using RGS under $m = 1, 2$ respectively.

4 Robust Generation Strategy (RGS)

4.1 Description of RGS

Because all messages from the population are stored in the bit's frequency, in order to increase robustness against uncertainty of GA, we calculate the bit's frequency string $F_{(t+m)}$ of the $t+m$ th generation as follows,

$$F_{t+m} = (f^1_{t+m}, \ldots, f^j_{t+m}, \ldots, f^L_{t+m}), \quad (m = 1, 2, 3, \ldots)$$

where

$$f^j_{t+m} = \frac{1}{m+1} \, (f^j_{\vec{X}'(t+m-1)} + \sum_{i=0}^{m-1} f^j_{t+i}) \qquad (8)$$

$f^j_{\vec{X}'(t+m-1)}$ means the j th bit's frequency of the population $\vec{X}'$ at the $t+m-1$ th generation. Eq.8 means the bit's frequency string at the $t+m$ th generation is determined not only by the bit's frequency string $F_{\vec{X}'(t+m-1)}$ but also by the bit's frequency string from the $t+m-1$ th to the t th generation. This method is named robust generation strategy(RGS).

4.2 Reason of Robustness

To investigate the effect of RGS, let us see a special case where $m = 1$. If $m = 1$, Eq.8 can be described as follows,

$$f^j_{t+1} = \frac{1}{2} \, (f^j_{\vec{X}'(t)} + f^j_t). \qquad (9)$$

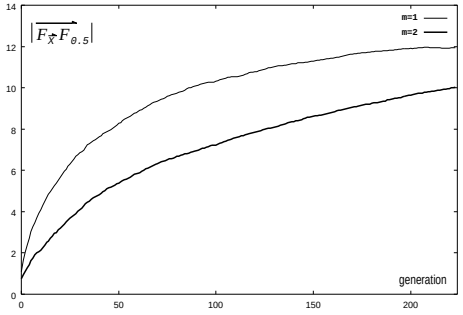


Fig. 7. Population concertrating rates by using RGS under $m = 1, 2$ respectively.

Eq.9 is a recursion formula and can be easily converted into as follows,

$$\begin{aligned} f^j_{t+1} &= \frac{1}{2} f^j_{\vec{X}'(t)} + \frac{1}{2^2} f^j_{\vec{X}'(t-1)} + \cdots + \frac{1}{2^t} f^j_{\vec{X}'(1)} \\ &+ \frac{1}{2^t} f^j_1 \end{aligned} \qquad (10)$$

According to Eq.10, we can see the effect of generations from the first to the t th on the $t+1$ th generation. This is reason why RSG can make GA search for solutions robustly against uncertainty than "conventional" GA. When $m \geq 2$, the relationship between f^j_{t+m} and $f^j_{\vec{X}'(t+m-1)}, \cdots, f^j_{\vec{X}'(1)}, f^j_1$ is a little difficult to be represented.

4.3 Experiments

The test function and all experiment's conditions are the same as the subsection 3.3. Fig.6 shows the simulation results under $m = 1, 2$ respectively. The upper part shows the mean function value while the lower part shows the standard deviation of function value.

According to the simulation results, we can get some following conclusions.

- From comparison between Fig.4 and Fig.6, we can see the mean fuction value and the standard deviation of Fig.6 are smaller than those of Fig.4. It means RSG can increase the searching ability and robustness of GA.
- Comparing $m = 1$ and $m = 2$, we can see the mean fuction value and the standard deviation of $m = 2$ are smaller than those of $m = 1$. It means the increase of m can make GA search for solutions more robustly.

From Fig.7 and Fig.5. we can see that the population concentrating rates of Fig.7 are slower than those of

Fig.5 while the concentrating rate of RGS under $m = 2$ is slower than that of RGS under $m = 1$.

5 Conclusion

In this paper, we focus on how the crossover and mutation work in GA by analyzing the variance of the bit's frequency and a new recombination method named RCBF is proposed. This method can make the population to distribute uniformly as large as possible without changing the bit's frequency string. It can increase the searching ability and robustness against uncertainty of GA, especially when the feasible solution space is very large. Based on RCBF, a new generation strategy named RGS is proposed where the $t+m$ th generation is determined not only by the $t+m-1$ th generation but also by generations from the $t+m-2$ th to the t th. Some experiments have clarified that RGS increases the searching ability and roubustness of GA as well.

References

[1] J. H. Holland, "Adaptation in Natural and Artificial Systems." Ann Arbor, MI: University of Michigan Press 1975.

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

[3] Alex Rogers, Adam Prügel-Bennett, "Genetic Drift in Genetic Algorithm Selection Schemes", IEEE Trans. Evol. Comput., vol. 3, no. 4. pp.298-303, 1999.

[4] J.E. Baker, " Adaptive selection methods for genetic algorithms", in Proceeding of the First International Conference on Genetic Algorithm, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 101-111 1985.

[5] Z. Michalewicz, "Genetic Algorithm + Data Structures = Evolution Programs", second, extended edition, Springer 1994

[6] X. Yao and Y.Liu, "Fast evolution strategies," IEEE Trans. Evol. Comput. Vol. 3, no. 2. pp.82-102, 1999.

[7] Yiu-Wing Leung and Yuping Wang, "An Orthogonal Genetic Algorithm with Quantization for Global Numerical Optimization," IEEE Trans. Evol. Comput. Vol. 5, no. 1. pp.41-53, 2001.

The Point of Point Crossover: Shuffling To Randomness

Anil Menon
Whizbang! Labs (East)
4616 Henry Street,
Pittsburgh, PA 15213
anilm@acm.org

Abstract

The action of point crossover is modeled as a random walk on a group, and convergence and rate results are established for the walk. Specifically, it is shown that there is a cut-off phenomenon in the rate at which the sample get randomized. As long as the number of crossover steps is less than a certain critical number, the total variation distance (with respect to the stationary distribution) is large, and remains essentially constant. But once the critical number has been crossed, the total variation distance goes to zero (at an exponential rate). The cut-off number of steps is of order of $O(lN \ln N)$ steps, where N is the sample size, and l is the length of the chromosome. Finally, it is shown by heuristic arguments as well as by simulations, that if a statistical criterion such as Kendall's W coefficient or the average Kendall's τ coefficient is used to measure randomness (rather than total variation distance), the sample can be said to be random (upto statistical significance) in $O(\ln N)$ steps, rather than $O(lN \ln N)$ steps. The properties of such criteria are characterized.

1 Introduction

The repeated application of point crossover on a finite set of chromosomes may be viewed as a random walk on a certain graph. The aim of this paper is to show that there is a cut-off phenomenon associated with a class of such "crossover walks." Roughly, the existence of a cut-off means that if the number of times point crossover applied in the crossover phase, n, is less than a certain critical number n^*, the sample remains "far" from stationarity, but for $n > n^*$, the sample becomes very "close" to stationarity.

There have been a variety of approaches to analyzing the role of point crossover, including (to list a few) hyperplane and schema analysis[13], dynamical systems models [5], and explicit Markov modeling [11]. However, despite the strong similarities between certain random walks and the crossover operator (for example, base swapping walks on matroids), not much work has been done to explore this connection, though there are a few outstanding exceptions [12]. In particular, the relationship between crossover walks and cut-off phenomena appears to have been overlooked.

Cut-off phenomena ("phase transitions") in random walks, especially those associated with walks on groups, have been intensely studied with great success in the last two decades [2]. The basic machinery behind these results draws upon deep results from the representation theory of groups. The techniques were first applied to study the effectiveness of various card shuffling operations, such as riffle shuffles, perfect shuffles and transposition shuffles. Intuitively, there is a great deal of similarity between shuffling sets of cards and the point crossover operator. In a sense, this paper formalizes this intuition. We eschew a too-rigorous presentation of results, and focus instead on heuristic arguments and simulations that will, hopefully, inspire a much more rigorous analysis.

The structure of the paper is as follows. In Section 2 the concept of a crossover walk is introduced. The question of its convergence is resolved by using techniques from the theory of doubly stochastic matrices. An analysis of the rate of convergence of the crossover walk is taken up in Section 3. In Section 4 it is argued that the traditional criterion used to measure the degree of randomness, namely, the variation distance, may be unnecessarily strict, and two alternate criteria are introduced. Section 5 presents simulations on the

behavior of these alternate measures, and Section 6 introduces an informal model to explain them.

Notation: S_n will denote the symmetric group on n symbols (the permutation group). All logarithms are to base e. Results drawn from external sources are referred to as "Propositions."

2 Crossover Walks

A chromosome is defined to be an element in Σ^l, where Σ is some finite alphabet. Any set of chromosomes defines a *sample*. The size of a sample is the number of chromosomes in it and two samples are distinct if they contain different numbers of any given chromosome in Σ^l. Let $\mathcal{S}_l(N)$ (or simply, $\mathcal{S}_l$) denote the set of samples of size N.

The k-point crossover operator $\times_k : \mathcal{S}_l \rightarrow \mathcal{S}_l$ maps one sample to another, and is defined as follows. Select a non-empty set of indices $I \subset \{1, 2, \ldots, l\}$ with respect to the "subset" measure Pr_k (explained below). Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_l)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_l)$ represent two chromosomes drawn uniformly from the input sample. Then, $\times_k(\alpha, \beta) = (\alpha', \beta')$ where for all $j \in I$, $\alpha'_j = \beta_j$ and $\beta'_j = \alpha_j$. For all $j \notin I$, $\alpha'_j = \alpha_j$ and $\beta'_j = \beta_j$. The resulting chromosomes α', β' are referred to the "children" of the "parent" chromosomes α, β.

The subset measure Pr_k is used to handle the differences between 1-point, 2-point crossover etc. All point crossover operators select a subset of indices from $\{1, \ldots, l\}$, and swap the corresponding alleles from the parents at these indices. 1-point crossover (1PTX) will always select subsets of the form $\{l\}$, $\{l-1, l\}$, $\{l-2, l-2, l\}$ etc. The symmetric version of 1PTX will also select subsets of the form $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$ etc. 2-point crossover (2PTX) will select subsets of the form $\{i, i+1, i+2, \ldots, i_k\}$ for $i \geq 1$ and $i_k \leq l$. In other words, each point crossover variant merely imposes an uniform probability measure on the set of all subsets of $\{1, 2, \ldots, l\}$. This probability measure is unique to each point crossover, and is denoted Pr_k.

The point crossover operator is usually seen as mapping chromosomal pairs to other chromosomal pairs, rather than one sample to another. While operationally there is no difference between the two views, modeling point crossover as taking samples to samples is analytically more convenient (at least for our purposes).

Define the *crossover graph* $G(S) = (V(S), E)$ where the node set $V(S) = \{v_1, \ldots, v_{|V|}\}$ is the set of all *distinct* samples that can be generated from the initial sample S by means of k-point crossover. $V(S)$ has at least one member, namely S, and $V(S)$ is a finite set because $\mathcal{S}_l$ is finite. Since k-point crossover replaces one pair with another, all samples in G have the same number of chromosomes.

Two nodes v_i and v_j in G are connected by an edge in the edge set E iff it is possible to generate v_j from v_i in *one* application of the $\times_k$ operator. In particular, every node v_i is connected to itself (self loop). The graph G is also connected (every node is reachable from every other node).

Let $\chi(S) = \{\times_k^j(S)\}_{j \geq 0}$ denote the *sequence* of samples that is generated by repeated applications of $\times_k$ on a sample S. Clearly, the sequence of samples in $\chi(S)$ then represents a random walk — the crossover walk — on the graph $G(S)$.

The two fundamental questions concerning the crossover walk on G are:

- Does the walk converge to a stationary distribution?
- If so, what is its rate of convergence?

The remainder of this section tackles the first question, and the rest of the paper considers the second.

2.1 Walk Convergence

The convergence to an stationary distribution can be established using a technique due to Feller [4, section XV.10]. As mentioned earlier, $\chi(S)$ represents the sequences of samples (nodes) encountered in walking the graph $G = (V, E)$. Define $P^t = (p_1(t), p_2(t), \ldots, p_{|V|}(t))$ where $p_i(t)$ is the probability that at time instant t, the walk finds itself at node v_i of the graph G. In other words, it is the probability that $\times_k^t(S) = v_i$. Initially, P^0 is a vector of all zeros except at one index r (say), corresponding to the fact that the walk starts at $S = v_r \in V$. From the definition of point crossover, the distributions at two successive instants of time are related by,

$$P^{t+1} = Q\, P^t \tag{1}$$

where $Q = [q_{i,j}]$ is a $|V| \times |V|$ sized matrix. The elements of Q can be assumed to be time independent, since the probability of moving to v_j given that the walk is at v_i should depend only the composition of the sample represented by v_i and v_j. Theorem 1 states the conditions under which the walk defined by Equation (1) converges.

Theorem 1 Let Q be the aperiodic, time independent transition matrix for a walk on the crossover graph $G(S) = (V(S), E)$. Then, Q is a doubly stochastic matrix, and the sequence $P^0, P^1, P^2, \ldots$, converges to the stationary distribution $P^\infty = (1/|V|, \ldots, 1/|V|)$. ∎

Proof: We first show that Q is a doubly stochastic matrix, that is, $\sum_{j=1}^{|V|} q_{i,j} = 1$ and $\sum_{i=1}^{|V|} q_{i,j} = 1$.

The matrix element $q_{i,j}$ is actually a conditional probability representing the probability of reaching v_j in one step, given that the walk is at v_i. By definition of conditional probabilities, $\sum_j q_{i,j} = 1$.

Observe that for every transition $(\alpha, \beta) \rightarrow (\alpha', \beta')$ produced by a k-point crossover operator, there corresponds a transition $(\alpha', \beta') \rightarrow (\alpha, \beta)$. In other words, if the k-point crossover operator transforms a sample $v_i \rightarrow v_j$, then it can also transform $v_j \rightarrow v_i$. Now, the expression $\sum_i q_{i,j}$ represents the probability that the vertex i can be reached from *some* vertex j. Since this can always be done, we conclude that $\sum_i q_{i,j} = 1$. Since Q is both column stochastic as well as row stochastic, Q is doubly stochastic.

From a standard result in Markov chain theory (for example, [4, section XV.7]) we know that if Q is persistent, irreducible and aperodic[1] then the sequence $P^0, P^1, P^2, \ldots$ converges to a stationary distribution. Because Q is doubly stochastic, it converges to the stationary distribution $P^\infty = (1/|V|, 1/|V|, \ldots, 1/|V|)$ [4, section XV.7, example 7(h)]. Q is persistent because it is doubly stochastic, and the construction of G guarantees that the walk on G is irreducible. From the aperiodicity, persistence and irreducibility of the walk, it follows that it converges to the stationary distribution P^∞. Q.E.D

The stationary distribution in Theorem 1 is related to but *not* the same as the linkage equilibrium distribution (which refers to the distribution of *chromsomes* in a sample randomized by crossover operations). Also, the assumption that Q is aperiodic is a trivial one, since any Markov chain can be redefined to be aperiodic [4, section XV.5].

Theorem 1 asserts that the repeated application of k-point crossover on a sample eventually randomizes it, and shows that point crossover belongs to a class of models known as quadratic dynamical systems [9]. The relationship between double stochasticity and point crossover leads to the Theorem 2. It shows that the class of Schur-convex functions are Lyapanuv functions for the crossover walk. A great deal is known about this class [8], and its functions occupy much real estate in mathematics[2]. A necessary and sufficient condition for a continuous function $\phi : R^n \rightarrow R$ to be Schur-convex is that it be symmetric ($\phi(x_1, \ldots, x_n) = \phi(x_{i_1}, \ldots, x_{i_n})$ and for any i, j, $(x_i - x_j)(\partial\phi/\partial x_i - \partial\phi/\partial x_j) \geq 0$.

Theorem 2 Let Q be the transition matrix for a walk on the crossover graph $G(S) = (V(S), E)$. Let $P^t = (p_1(t), p_2(t), \ldots, p_{|V|}(t))$ where $p_i(t)$ is the probability that at time $t \geq 0$, the walk finds itself at node v_i. If $F : R^{|V|} \rightarrow R$ is a Schur-convex function, then for all $t \geq 0$, $F(P(t+1) \leq F(P(t))$. ∎

Proof: The theorem is an immediate consequence of three facts: (1) $P(t+1) = QP(t)$, (2) Q is doubly stochastic (Theorem 1) and (3) the Hardy, Littlewood, Polya theorem (see [8, chap. 2, B.2] and [8, chap. 3, A.1]. Q.E.D.

3 Rate of Convergence

Any analysis of the convergence rate of a crossover walk depends on the "intrinsic" aspects of the walk such as the transition probabilities, and the exact k-point crossover used. In particular, it depends on the structure of the graph G. Since the structure of the graph is determined by the initial sample, the convergence rate of the walk is dependent on it.

The dependency of the walk on the initial sample complicates matters, perhaps unnecessarily so. Suppose initial sample consists of identical chromosomes. Clearly, point crossover is not going to change the composition of the sample. In this scenario, $\chi(S) = \{S, S, \ldots\}$ and $G(S) = (\{S\}, E)$ where E consists of a single self-loop. On the other hand, consider the effect of point crossover on a sample drawn randomly from Σ^l. In this case too, point crossover has no effect since the sample is already randomized, and a walk on G is essentially a walk on a random graph. It is difficult to study the general walk on G, because G can take on so many different "shapes" depending on how the initial sample was set up.

An analogy might make this idea clearer. Suppose one wished to analyze the shuffling of a pack of cards (pos-

[1] A Markov chain $Q = [q_{i,j}]$ is said to be *persistent* if it is certain that the chain starting from a state v_i will eventually return to v_i. The chain is said to be *irreducible* iff every state v_i can be reached from any other state v_j. Q is said to be *aperiodic* if $q^t_{i,i} \neq 0$ for any $t > 1$.

[2] For example, the Shannon Entropy function is a Schur-concave function (that is, negentropy is Schur-convex). So are almost all of the popular diversity metrics, such as sample variance and the Gini coefficient. It is *not* necessary that a function be continuous in order for it to be Schur-convex.

sibly incomplete), where the shuffle operation consists of transposing (with some probability) pairs of cards drawn from the pack w.r.t some probability measure. As stated, the problem is hard to study because it mixes the critical issue (the effect of random transposition) with the less important ones (possibly incomplete packs, unknown initial card distribution, transposition frequency, a measure on drawing cards etc). It is for this reason that most random transposition models assume an initially sorted, complete pack, where every shuffling step results in a transposition (unless the same card is picked twice).

Of course, the decision as to which factors are important and which ones are not, depends on the problem one is interested in. For example, if the problem is to study how point crossover "undoes" the effect of proportional selection, then the dependency on the initial sample has to be taken into account. But if the problem is (say) to prove that repeated applications of point crossover leads to linkage equilibrium, then the specifics of the initial sample is not too important (as Theorem 1 demonstrates).

What is needed is a *reference sample* against which the effectiveness (as measured by rates of convergence) of various crossover operators can be tested. In other words, a reference sample will enable the distinction between "what crossover does" from "what crossover is applied to."

The definition of the point crossover operator suggests that its action is roughly analogous to a shuffling operation on sets of decks of cards. Accordingly, a good reference sample to study its convergence rates should be an array of permutations. Specifically, consider an array of numbers (N rows and l columns), arranged as follows:

$$S = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2 & \cdots & 2 \\ \vdots & \vdots & \cdots & \vdots \\ N & N & \cdots & N \end{pmatrix} \tag{2}$$

Each *row* in the array is interpreted as a "permutation" chromosome of length l. Upon applying k-point crossover on S, the columns of the array will tend to get randomized. For example, one such sample is,

$$S' = \begin{pmatrix} 1 & 2 & \cdots & 2 \\ 2 & 1 & \cdots & 1 \\ \vdots & \vdots & \cdots & \vdots \\ N & N & \cdots & N \end{pmatrix} \tag{3}$$

Of course, k-point crossover only disarrays a column, and does not change the allelic composition, at any step of the walk. Hence, the sample can be represented as a vector of permutations $(\eta_1, \ldots, \eta_l)$, with $\eta_j \in S_N$ where S_N is the permutation group on N symbols. Initially, $S \equiv ((1,2,\ldots,N),\ldots,(1,2,\ldots,N))$. The action of k-point crossover at any step consists of selecting at random a non-empty, proper subset of components from $(\eta_1, \ldots, \eta_l)$ and applying a random transposition drawn from the permutation group S_N to each of those components. For example the move from $S \rightarrow S'$ involves selecting the components 2 through l and applying the transposition $(1,2)$ on each of those components.

In other words, the underlying crossover graph has $(N!)^l$ nodes. An edge connects two nodes $v_i = (\sigma_1, \ldots, \sigma_l)$ and $v_j = (\eta_1, \ldots, \eta_l)$ iff there exists a transposition $\tau \in S_n$ and a subset $I \subset \{1, 2, \ldots, l\}$, such that,

$$\eta_j = \begin{cases} \tau(\sigma_j) & \text{if } \ j \in I, \\ \sigma_j & \text{otherwise.} \end{cases} \tag{4}$$

The effect of k-point crossover on the N "permutation" chromosomes is thus identical to a nearest-neighbor random walk on the nodes of the crossover graph.

Of course, real GAs do not typically operate on permutation strings, so the relevance of the above reference sample may be in question. It would appear however that the walk on $G(N,l)$ provides a good test case for analytic techniques, is related to active areas of research in probability theory, and focuses on point crossover's central feature, namely, its tendency to "shuffle" a sample's alleles. Furthermore, there are techniques to "lift" walks on symmetric groups to walks on the hypercube [2, pp. 19-20], so results may transfer as well.

3.1 Walks on $G(N,l)$

Since the structure of this graph is completely determined by the parameters N and l, it will be denoted $G(N,l)$ rather than the usual $G(S)$. It is not hard to show that $G(N,l)$ is a regular graph, where each node is connected to $d = (2^l - 1)\binom{N}{2}$ other nodes.

The analysis of the convergence rate of the walk on $G(N,l)$ depends on the criterion used to measure the "degree of randomness." One criterion popular in models of random walk on groups [2, chap. 3B] is the variation distance of two discrete distributions P and P' defined by,

$$||P - P'|| = \frac{1}{2}\sum_{i=1}^{n} |p_i - p'_i| \tag{5}$$

In particular, if for any specified $\epsilon > 0$, there exists a t_c such that for all $t > t_c$, $||P^t - P^\infty|| < \epsilon$, then

the walk is said to converge to P^∞ w.r.t the variation distance.

A formal analysis of the rate of convergence of $||P^t - P^\infty||$ is complicated and involves the machinery of the representation theory of the symmetric group. Also, in this paper, the quantity of interest is not the variation distance but a different function (described in the next section). Hence, we will only present heuristic arguments for the rate of convergence of the total variation distance on $G(N,l)$.

Consider the case when $l = 1$. The problem then reduces to the random walk induced by the action of random transpositions on the symmetric group S_N. P. Diaconis and M. Shahshahani proved the following result regarding such walks [3].

Proposition 1 Let $k = \frac{1}{2}N \ln N + cN$, where $c > 0$. Then, there exists an universal constant a such that, $||P^t - P^\infty|| \leq a \exp(-2c)$. Correspondingly, let $k = \frac{1}{2}N \ln N - cN$, where $c > 0$. Then, there exists an universal constant b such that, $||P^t - P^\infty|| \geq 1 - b \exp(-2c)$. ∎

Proposition 1 shows that $k = \frac{1}{2}N \ln N + cN$ are sufficient for the variation distance to become "small". Conversely, $k = \frac{1}{2}N \ln N - cN$ are also necessary. This is the celebrated "cut-off' phenomenon, the value of $||P^t - P^\infty||$ is large for $k < \frac{1}{2}N \ln N - cN$, but is small after $k \geq \frac{1}{2}N \ln N + cN$.

This case provides a lower bound on the convergence rate of the random walk on $G(N,l)$. On the other hand, it can be shown using several different ways[3] that $\frac{Nl}{2} \ln N$ steps is an upper bound on the number of steps necessary to achieve stationarity

In summary, it takes at most $O(Nl \ln N)$ k-point crossover operations to randomize an array of N chromosomes of length l, where the starting sample is given by Equation (2). If the reference sample consists of N binary chromosomes each of length l, rather than permutation strings, the conclusions of the analysis of the walk on $G(N,l)$ does not fundamentally change. For example, consider the crossover walk on a reference sample of binary chromosomes where half the sample is initially "all 0"' chromosomes, and the other half are "all 1" chromosomes. This walk can be shown to be closely related to the Bernoulli-Laplace urn model [2, pp. 56-58], for which the cutoff number of steps again turns out to be $O(Nl \ln N)$.

[3]Wald's principle offers one route. Another option is to note that a walk on $G(N,l)$ can be described as a walk on the Cartesian product of the transposition graph of the symmetric group, and then use Chung's results [1, pp. 36-41].

But are such "$O(N \ln N)$" results of any practical use? In most GAs, the sample size N, and the chromosome length l, are both quite large (typically). For $N = 500$ and $l = 30$, the above result would indicate that the number of crossover steps is of the order of $100,000$ steps. Thus, the number of crossover steps required to randomize the sample is quite large, and it would appear that randomization of the sample in the point crossover phase never happens in real GA deployments.

However, in the next section it will be argued that the large number of crossover steps required to randomize the walk on $G(N,l)$ (and by association, walks on general samples), is an artifact of the variation distance criterion. The rate of convergence of two alternative criterion, the Kendall's average τ coefficient, and Kendall's W coefficient, give a very different picture on the minimal number of steps required to randomize the walk on $G(N,l)$.

4 Convergence Criteria

As far as convergence is concerned, the exact norm used to measure the distance between two distributions is not of great importance, since norms are (topologically) equivalent (so convergence w.r.t one norm implies convergence w.r.t another). But for bounds on convergence *rates*, the choice of the norm is very important [14].

The variation distance may be inappropriate in some natural context. Suppose one is given a set of decks, where each deck is arranged in some manner (not necessarily sorted). It is now required to be determined whether the cards in the decks are randomly ordered or not. The variation distance is not a very meaningful measure in this case. The statistical solution is to compute some ranking statistic on the card arrangements, and see if the null hypothesis (card are randomly arranged in each deck) can be rejected. Clearly, this idea can be also applied to each sample produced in the random walk on $G(N,l)$. Statistical tests have been evolved to test for randomness (upto specified levels of significance). It makes sense to use them to test whether the sample produced by crossover at any stage passes these tests. If it does, then we have a rigorous basis for a stopping rule.

In this case, the sample consists of permutations, and it is natural to study the change in rank-based concordance measures as a function of the stage in the random walk on the crossover graph. Two such measures will now be considered. The first, Kendall's W coefficient, is a measure of ranking concordance and

the second, Kendall's average tau coefficient, is usually interpreted as a measure of disarray of in a set of permutations.

Consider l judges ranking N objects. Each judge assigns a distinct rank to each object. The rankings can be arranged in an array of the type shown in Equation (2), where each column represents a ranking, and the i^{th} row reflects how each judge ranks the i^{th} object. Let s_{ij} denote the ranking of the i^{th} object by the j^{th} judge.

If all the columns are identical as in Equation (2), it indicates complete concordance between the judges. A standard measure of concordance is *Kendall's W coefficient* [6, chap. 6] defined as follows:

$$W = \frac{12\,D}{l^2 N(N^2-1)}, \tag{6}$$

$$\text{where,} \quad D = \sum_{i=1}^{n}\Big(\sum_{j=1}^{l} s_{i,j} - \mu\Big)^2, \tag{7}$$

$$\text{and,} \quad \mu = \frac{l(N+1)}{2}. \tag{8}$$

The idea is to compute for each object, the sum of the ranks assigned to it by the l judges. The sum of the squares of the deviation of each sum from the expected value μ then gives D. Kendall's W is the ratio of D with the maximum possible value. Kendall's W coefficient is widely used to measure the agreement in l rankings of a common set of objects [6]. W always lies between 0 and 1, with 1 indicating complete agreement between the rankings of the l judges.

The second ranking statistic is the *average Kendall coefficient* [2, chap. 6]. Let σ and π be any two permutations in S_N (permutation group on N symbols). Let $K(\sigma,\pi)$ $(= K(\pi,\sigma))$ be the number of adjacent transpositions required to convert the permutation σ^{-1} to π^{-1}. Kendall's τ coefficient for the pair σ and π is defined by,

$$\tau(\sigma,\pi) = 1 - \frac{4K(\sigma,\pi)}{N(N-1)}. \tag{9}$$

The τ coefficient lies between -1 and 1 (inclusive) and behaves likes a correlation coefficient. When $\tau = 1$, the permutations are identical, and when $\tau = -1$, $\sigma = \pi^{-1}$. The *average Kendall coefficient* for a set of permutations $(\sigma_1, \sigma_2, \ldots, \sigma_l)$ is given by,

$$\bar{\tau} = \frac{2\sum_{i,j=1}^{l}\tau(\sigma_i,\sigma_j)}{l(l-1)}. \tag{10}$$

Both these measures are designed to measure the degree of disarray in a sample of permutations, and their asymptotic behavior is well understood[4][6, chap. 6]. This enables their practical use in statistical significance tests.

The next section studies the change in these ranking statistics as a function of the crossover walk on $G(N,l)$ for various values of N and l. The associated graphs not only show the existence of cut-off behavior in these functions for crossover walks, but also show that, remarkably, the number of steps required to achieve randomness (upto statistical significance) is of the order of $O(\ln N)$ rather than $O(lN\ln N)$.

5 Simulations

Here the behavior of the crossover walk on the graph $G(N,l)$ for various values of N and l are studied. The basic procedure for setting up the simulations was to start with the ordered $N \times l$ array shown in Equation (2). Then, k-point crossover was repeatedly applied (usually for 100,000 steps). Each application of the operator corresponds to a step on the graph $G(N,l)$. After applying the operator, the values of the average τ and/or Kendall's W-coefficient for the sample are computed. These values are then plotted against the *logarithm* of the step number [5]. The shape of the curve, its critical points and sensitivity to the three independent variables, namely, N, l and k, are the main topics of interest. Here, only the results for fixed l and k but varying N are presented. It is worth mentioning however, that all scenarios show the existence of the cut-off phenomena, though the exact point at which cut-off happens, changes as the dependent variables are changed.

Figure 2 and Figure 3 show the plots obtained by sampling the values of Kendall's W coefficient and Kendall's average τ after every 1PTX step of the random walk on $G(N,l)$ for $N = 50, 100, 150, 200$ and $l = 15$. The curves become smoother for larger values of N, but in general the behavior is relatively insensitive to changes in values of N. Consider the point at which the Kendall's W coefficient falls below 0.5. For $N = 50, 100, 150, 200$, this happens (roughly) at $\exp(3.2) \approx 25$, $\exp(4.8) \approx 122$, $\exp(5.2) \approx 181$ and

[4] For example, the asymptotic distribution of Friedman's function $\chi_r^2 = l(N-1)W$ can be shown to be approximately χ^2 with $N-1$ degrees of freedom for large l. On the other hand, the average τ can be shown to distributed normally.

[5] The cutoff phenomenon implied by the sigmoid growth curve disappears if the ranking statistics is plotted directly against the step number. This may be one reason why the cut-off phenomena in ranking statistics for random walk models appears to have escaped the attention of probability theorists.

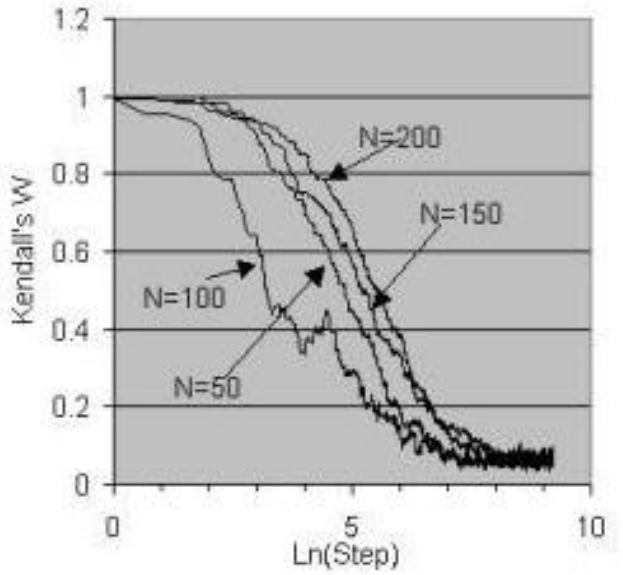


Figure 2: Kendall's W versus Ln(Crossover Step) for various population sizes

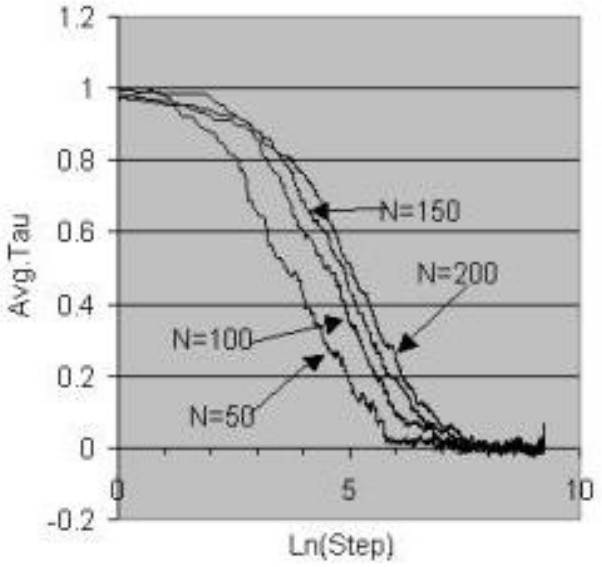


Figure 3: Kendall's Average Tau versus Ln(Crossover Step) for various population sizes

$\exp(5.57) \approx 262$ number of steps, respectively. This may seem like a significant change, but if considers the *ratio* of the number of steps (to get to the half-way point) to the sample size, the ratios are roughly constant.

The most significant aspect of these figures is how soon the cut-off point manifests itself. For $N = 200$, by about $\exp(2)$ to $\exp(3)$ number of steps, the Kendall's W starts to fall sharply, and by about $\exp(7)$ to $\exp(8)$ number of steps, it reaches its equilibrium value. Thus a few hundred applications of point crossover does have a significant impact on the value of Kendall's W coefficient for the walk on $G(N, l)$. Similar comments hold for the average τ.

6 Logistic Models

A rigorous analysis of the behavior of Kendall's W coefficient, or the average τ coefficient is likely to be very complicated. Yet, the curves are so simple in their shape that it is very tempting to believe that an equally simple explanation must be available.

In this section, an explanation based on an population growth model will be developed; it is simplistic, but the basic idea is very general and holds much promise (e.g. [7, 10]).

The problem is to model the change in a statistic Y w.r.t. the step number. Suppose it was the case that there were two kinds of events that affected the growth of Y. The "good" events cause it to increase, but the "bad" events cause it to decrease. It is also given that if only bad events happen, then the relative change in Y is inversely (directly) proportional to the relative change in r. One way to model this is,

$$\frac{\Delta Y}{Y} = \alpha(Y)\frac{\Delta t}{t}, \quad \alpha > 0, \tag{11}$$

$$\frac{\Delta Y}{Y} = -\beta(Y)\frac{\Delta t}{t}, \quad \beta > 0. \tag{12}$$

The linear relationship has been setup not between Y and t, but between the relative growths $\Delta Y/Y$ and $\Delta t/t$. The basic reason for this is that cutoff phenomena persist under scaling changes, that is, cannot be removed by ratio transformations of the dependent and independent variables. If the differential equation we are constructing is to exhibit cut-off phenomenon, then it has to be invariant under ratio transformations as well. Equations (11) and (12) have this property.

The quantities α and β have been marked as a function of Y but not time. The reason for this is that the dynamics of *any* two variables U and V can be related vacuously by a "constant" that varies with respect to U and V. To prevent this, the proportionality constants can depend at most on Y. The dependency on Y models the fact that Y, being a ranking statistic, cannot grow ceaselessly. Since it is a ranking statistic, it takes on at a finite number of values (there are only a finite number of rankings, and each ranking corresponds to one value for the statistic).

α and β are duals to each other. Assume without loss of generality that $\alpha + \beta = 1$ and $\alpha \leq 1$ (if they are not, the equations can always be rescaled to make it so). Then, for some function $g(Y)$, the functions α and β can be expressed as,

$$\alpha(Y) = 1 - \frac{g(Y)}{K}, \quad (13)$$

$$\beta(Y) = \frac{g(Y)}{K}. \quad (14)$$

where K is large enough to make $\alpha, \beta \leq 1$. Putting the above equations together,

$$\frac{\Delta Y}{Y} = \frac{\Delta t}{t}(1 - 2\frac{g(Y)}{K}). \quad (15)$$

Passing to the limit implies,

$$\frac{dy}{dt} = \frac{y}{t}(1 - 2\frac{g(y)}{K}). \quad (16)$$

To "draw" the above curve with respect to the logarithmic axis, set $t = \ln x$. Consequently,

$$\frac{dy}{dx} = y(1 - 2\frac{g(y)}{K}). \quad (17)$$

Equation 17 produces a sigmoid curve under very mild restrictions on the function $g(y)$. The case $g(y) = y$ leads it to the classic Verhulst-Pearl equation of (sigmoid) growth.

The assumptions behind this heuristic argument are minimal. All that is required is that Y be density limited, its growth has to be explainable by a two factor model (good events/bad events), and $(\Delta Y/Y) \propto (\Delta t/t)$.

Consider Kendall's average τ coefficient. Since it is a ranking statistic, it cannot grow without bounds. Every application of point crossover splits the permutations in the sample into two groups, namely, those that got affected by the crossover, and those that didn't. The τ coefficient of each pair changes only linearly with every crossover step[6]. The τ coefficients within each group do not change, but the inter-group τ-coefficients do change. The extent of that change is proportional to the product of the relative sizes of the two groups, and hence the *log* of the changes is linearly proportional to the logs of the relative sizes.

The growth in the average τ is also driven by a two-factor model, because the change in τ is driven by a two factor-model. A "good" change consist of a transposition which reduces the disarray, and hence increases the value of τ. Correspondingly, a bad transposition is one which increases the disarray and reduces the value of τ.

[6]Recall that the τ-coefficient of a pair of permutations is an affine function of the number of adjacent transpositions needed to transform one permutation to the other.

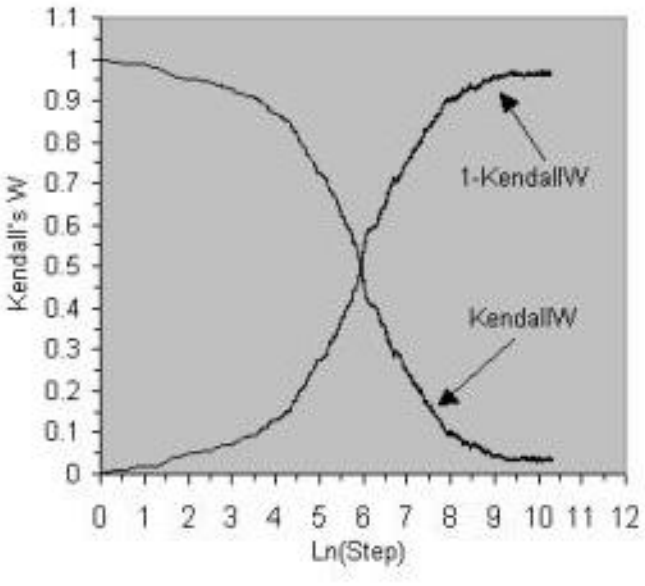


Figure 4: Kendall's W versus Ln(Crossover Step) for 1PTX walk on $G(300, 30)$ (sample of 300 chromosomes each of length 30)

Similar arguments can be made for Kendall's W coefficient, though the details are a lot more tedious. In any event, these arguments are meant to be suggestive of the possibilities of an alternative to the currently popular group-theoretic approaches.

Finally, Figure 4 shows a plot of Kendall's W for repeated applications of 1PTX on a population of 300 permutations each of length 30. Clearly, even for these small population sizes and non-trivial chromosome lengths, the sigmoid growth curve is obtained. Notice that by approximately $6\log(N)$ steps, the measure falls to its mid value (0.5).

7 Conclusion

What does point crossover do? The results of the paper formalize the intuition that repeated applications of point crossover "shuffles" a sample's alleles. The formalization was achieved by modeling the action of point crossover as a random walk on the crossover graph. Two aspects of this walk were studied. First, it was demonstrated that this walk is characterized by a homogeneous doubly stochastic Markov chain and hence may be shown to converge to a stationary distri-

bution. Second, the rate of convergence was analyzed, and it was shown that there is a cut-off phenomenon in the rate at which the original sample get randomized by the repeated action of point crossover. As long as the number of crossover steps is less than a certain critical number, the total variation distance (with respect to the stationary distribution) is large, and remains essentially constant. But once the critical number has been crossed, the total variation distance goes to zero (at an exponential rate). The cut-off number of steps is of order of $O(lN \ln N)$ steps, where N is the sample size, and l is the length of the chromosome. If different metrics are considered, say, Kendall's W or average τ coefficient, then simulation indicate that cut-off occurs at $O(N \ln N)$ rather than $O(lN \ln N)$. A heuristic explanation based on population arguments was provided for the general sigmoid nature of these curves. The existence of the cut-off suggests that point crossover is something of an all-or-nothing randomization operator. Apply it for more than the cut-off number, and the sample is rapidly randomized. Apply it for less, and as far as randomization is concerned, the sample remains far from random. Whether such phase transitions exist for other crossover operators remains an open question.

8 Acknowledgements

I would like to thank the two reviewers whose comments helped to make this a better paper. I would also like to thank Dennis Wakefield for his assistance in helping me meet the submission deadline, and Saras Sarasvathy for her general support and encouragement.

References

[1] Fan R. K. Chung. *Spectral Graph Theory*. Number 92 in Regional Conference Series in Mathematics. American Mathematical Society, Providence, Rhode Island, 1997.

[2] P. Diaconis. *Group representations in probability and statistics*. IMS Lecture Notes—Monograph Series, Hayward, CA, 1988.

[3] P. Diaconis and M. Shahshahani. Generating a random permutation with random transpositions. *Z. Wahrscheinlichkeitstheorie Verw. Gebiete*, 57:159–179, 1981.

[4] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume I. Wiley Eastern, New York, Third edition, 1968.

[5] S. Forrest and G. Mayer-Kress. Genetic algorithms, nonlinear dynamical systems, and global stability models. In L. Davis, editor, *The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY., 1991.

[6] M Kendall and D. Gibbons. *Rank correlation methods*. Edward Arnold, London, fifth edition, 1990.

[7] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, 7:49–58, 1970.

[8] A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and its Applications*. Academic Press, New York, 1979.

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

[10] M. Mitzenmacher. Studying balanced allocations with differential equations. Technical Report 1997-024, SRC Technical Note, Digital Systems Research Center, CA, 1997.

[11] E. A. Nix and D. M. Vose. Modeling genetic algorithms with markov chains. *Annals of Math. and Artificial Intelligence*, 5:79–88, 1992.

[12] Y. Rabani, Y. Rabinovich, and A. Sinclair. A computational view of population genetics. *Random Structures and Algorithms*, 12(4):313–334, 1998.

[13] W. M. Spears and K. DeJong. An analysis of multi-point crossover. In *Foundations of Genetic Algorithms*, volume 1, pages 301–215. Morgan Kaufman, 1991.

[14] Lloyd N. Trefethen and Lloyd M. Trefethen. How many shuffles to randomize a deck of cards? *Proc. Royal Society London, Series A*, 456:2561–2568, 2000.

A Comparison of Memetic Recombination Operators for the Traveling Salesman Problem

Peter Merz
University of Tübingen
Department of Computer Science - WSI-RA
Sand 1, D-72076 Tübingen, Germany
peter.merz@ieee.org

Abstract

Several memetic algorithms (MAs) – evolutionary algorithms incorporating local search – have been proposed for the traveling salesman problem (TSP). Much effort has been spent to develop recombination operators for MAs which aim to exploit problem characteristics to achieve a highly effective search.

In this paper, several recombination operators for the TSP are compared. For the purpose of identifying the important properties of a recombination operator, a new generic recombination operator (GX) is proposed which is comprised of four phases. These phases can be controlled by parameters reflecting the most important properties of recombination operators. It is shown that GX recombination is superior to MPX and DPX when all common edges are preserved in the offspring.

1 INTRODUCTION

The traveling salesman problem (TSP) is one of the best-known combinatorial optimization problems. It can be stated as follows: Given n cities and the geographical distance between all pairs of these cities, the task is to find the shortest closed tour in which each city is visited exactly once. From a graph theoretical point of view, this is equivalent to finding the shortest Hamiltonian cycle in a complete graph.

The TSP has been widely used as a problem for testing new heuristic algorithms and general purpose optimization techniques. In particular, several evolutionary algorithms have been proposed to tackle this $\mathcal{NP}$-hard problem. Simple evolutionary algorithms have been shown to be ineffective in finding near optimum solutions [9]. Therefore, several researchers incorporated local search into an evolutionary framework such that all individuals in the population are local optima, leading to highly effective algorithms, known as genetic local search or memetic algorithms (MAs)[23, 24]. Memetic algorithms have been shown to be among the best heuristics for the TSP [18, 13, 17, 12, 10, 1]. Other important aspects of MAs not covered in this paper include spatial population structures [7], MA theory [29], and self-adaptation [14].

In this paper, memetic algorithms for the TSP are studied by concentrating on the most important part of the evolutionary framework – the recombination of solutions. A new generic greedy recombination operator is introduced to the study three important aspects of TSP tour recombination: the inheritance of common edges to both parents, the insertion of new edges, and the inheritance of edges found in just one of the parents. The greedy operator can be controlled by three parameters in respect to these aspects. Several parameter settings of GX are compared with maximally preserving crossover (MPX) [25, 8] and distance preserving crossover (DPX) [6, 18]. The experiments show the importance of inheriting common edges and provide a meaningful choice of the remaining two parameters. The effectiveness of the MA with GX is demonstrated on several instances form TSPLIB [28].

The paper is organized as follows. In section 2, the memetic algorithm framework used in this paper, as well as the new greedy recombination operator is introduced. In section 3, a comparison of memetic recombination operators is performed on selected TSP instances. And results are presented of the MA using GX on 15 TSP instances. Section 4 concludes the paper and outlines areas for future research.

2 MEMETIC ALGORITHMS FOR THE TSP

Although there are other effective evolutionary algorithms incorporating sophisticated problem dependent procedures such as Nagata and Kobayashi's [26] evolutionary algorithm with *edge assembly crossover*, MAs provide a general framework for hybrid algorithms that can be applied to other combinatorial problems such as the quadratic assignment problem [20], the binary quadratic programming problem [19], and graph bipartitioning [21].

```
procedure MA;
    begin
        initialize population P;
        foreach i ∈ P do i := Local-Search(i);
        repeat
            for i := 1 to #recombinations do
                select two parents i_a, i_b ∈ P randomly;
                i_c := Recombine(i_a, i_b);
                i_c := Local-Search(i_c);
                add individual i_c to P;
            endfor;
            for i := 1 to #mutations do
                select parent i_a ∈ P randomly;
                i_c := Mutate(i_a);
                i_c := Local-Search(i_c);
                add individual i_c to P;
            endfor;
            P := select(P);
            if P converged then
                foreach i ∈ P\{best} do
                    i := Local-Search(Mutate(i));
            endif
        until terminate=true;
    end;
```

Figure 1: The Memetic Algorithm

MAs for the TSP are similar to other evolutionary algorithms: a population of locally optimal solutions is evolved over time by applying evolutionary variation operators (mutation and recombination operators), and by selection of the best individuals from the pool of parents and offspring. The pseudo code for the MAs used in this contribution is shown in Fig. 1. To ensure that the individuals in the population are local optima, after each application an evolutionary variation operator, local search is applied. This includes the initialization phase of the population in which solutions are constructed from scratch: A local search procedure is applied to these solutions so that even the first generation consists exclusively of local optima.

The problem-specific parts of the algorithm comprise initialization, local search, and the evolutionary variation operators: recombination and mutation. In comparison to other evolutionary algorithms, the role of mutation and recombination is different. Firstly, mutation and recombination are performed independently from each other. Secondly, the phenotypic changes caused by the variation operators must be large enough to reach the basin of attraction of new local optima, since local search is always applied after mutation or recombination.

2.1 Initialization and Local Search

To initialize the population of the MA, TSP tours have to be generated either randomly or by a randomized tour construction heuristic such as nearest neighbor or the greedy heuristic [11]. After the generation of feasible tours, a local search is applied.

The most effective local search procedures for the TSP are *2-opt*, *3-opt*, and the Lin-Kernighan (LK) heuristic [15, 11]. These heuristics exchange 2, 3, or a variable number of edges in each iteration, respectively. Generally, the stronger the local search used the better the performance of the MA. Therefore, the Lin-Kernighan heuristic has been used in [6, 5, 18].

2.2 Recombination Operators

During recombination, a new offspring is generated by copying edges from the parents. However, the TSP tour constraints have to be obeyed: each node (city) is connected with exactly two other nodes via two edges and the tour is required to have only one cycle. These constraints are hard to obey, hence many proposed recombination operators introduce foreign edges which are not contained in one of the parents to meet the constraints. These foreign edges can be considered as *implicit mutations*, and have a high impact on the performance of EAs for the TSP [16], since they can be very long, destroying the benefit of combining the short edges from the parents.

2.2.1 Properties of Recombination Operators

The use of local search after the application of a recombination operator – as is the case in memetic algorithms – can compensate for the disruptive effects of implicit mutations. In some cases, implicit mutations have a positive effect on the performance of the local search, and in some situations they have not. Thus, it is important that implicit mutations can be controlled in some way. Besides the number of foreign edges introduced during recombination, another aspect appears to be important: which edges are in-

herited from the parents and which are not. More formally, recombination operators can be classified according to Radcliffe and Surry [27] as

Respectful: The alleles that are identical in both parents are preserved in the offspring, i.e. all edges found in both parent tours (common edges) are found in the offspring tour

Assorting: The offspring contain only alleles from either one of the parents, i.e. all edges in the child tour are found in at least one of the parent tours, thus no implicit mutation occurs

While respectful recombination can be easily achieved by a recombination operator for the TSP, assorting recombination is hardly accomplished. Note, that for binary representations a respectful recombination is also assorting.

2.2.2 MPX and DPX

Although there are many recombination operators proposed for the TSP, we concentrate on those especially useful in combination with local search und thus in a memetic framework. Other recombination operators such as the *edge recombination operator family* [16, 3] or the *edge assembly crossover* [26] are aimed at preserving edges without additional local search. These operators are inferior to other more disruptive operators if local search is used [3, 17].

In the MPX proposed in [8] a sub-path between two randomly chosen crossover points is copied from the first parent to the offspring. The first crossover point is chosen to be at an edge not contained in the second parent. The partial tour is extended by copying edges from the second or first parent afterwards. If no parental edge can be included a foreign edge is introduced to maintain feasibility. To a high extent, edges from the parents are retained. This operator does not guarantee to be respectful.

The DPX proposed in [6, 5] is an operator that is only useful in combination with local search. In contrast to MPX or other recombination operators such as the edge recombination operators [30], it forces the inclusion of foreign edges in the offspring instead of preventing it.

DPX tries to generate an offspring that has equal distance to both of its parents, i.e., its aim is to achieve that the three distances between offspring and parent 1, offspring and parent 2, and parent 1 and parent 2 are identical. It works in two phases: (1) all common edges are copied to the offspring, and (2) the tour fragments present in the offspring are reconnected based on a nearest neighbor algorithm where edges contained in one of the parents are not considered.

2.2.3 The Generic Greedy Recombination Operator

A new recombination operator is proposed in the following that utilizes the greedy construction scheme of the greedy heuristic [11]. The *generic greedy recombination operator (GX)* consists of four phases:

Phase I: (*common edges*)
In the first phase, some or all edges contained in both parents are copied to the offspring tour.

Phase II: (*new edges*)
In the second phase, new short edges are added to the offspring that are not contained in one of the parents. These edges are selected randomly among the shortest edges emanating from each node. These edges are with high probability contained in (near) optimum solutions and are thus good candidates for edges in improved tours.

Phase III: (*non-common edges*)
In a third phase, edges are copied from the parents by making greedy choices. Edges are inserted in order of increasing length, and only candidate edges are considered, i.e., edges that violate the TSP constraints.

Phase IV: (*remaining edges*)
In the fourth and last phase, further edges are included in order of increasing length until the child consists of n edges and is thus a feasible TSP tour.

All greedy choices in the fourth step are randomized by selected the shortest remaining edge with a probability of 0.66 and the second shortest edge with a probability of 0.33.

The GX operator has three parameters: the common edges inheritance rate (cRate) that determines the probability that a common edge is added to the child and is thus a control parameter for the first phase. With a rate of 1.0, respectful recombination is achieved, all other rates lead to disrespectful recombination. The second phase is controlled by the new edges insertion rate (nRate) that determines the number of new edges to include. A rate of 0.5, for example, determines that half of the remaining edges to insert after phase one are new edges that are short but not contained in one of the parent solutions. The maximum number of edges to inherit from the parents is determined by the inheritance rate (iRate). In the

last phase, allowed edges in increasing length are chosen that may or may not be found in the parents. For am more detailed explanation see [17].

2.3 The Mutation Operator

Simple mutation operators are not suited for use in MAs, since subsequently applied local search procedures will usually revert the changes made. For example, the inversion operator randomly exchanging two edges is ineffective when *2-opt*, *3-opt* or LK local search is used. Therefore, in MAs alternative mutation operators are required.

The non-sequential four change (NS4) is an edge exchange involving four edges [15]. It is especially useful in connection with the LK heuristic. Since LK only performs sequential exchanges, it cannot reverse a non-sequential four change in one iteration. The NS4 is used in the iterated Lin-Kernighan heuristic [11], which is known to be very effective.

2.4 Selection and Restarts

In this work, a single panmictic population structure is used. Thus selection utilized in the memetic algorithms is a global selection strategy and similar to the selection in the $(\mu + \lambda)$-ES (*Evolution Strategy*): The new population is derived by selecting the best individuals out of the pool of parents and children. Duplicates are eliminated such that a solution is contained no more than once in the population.

Due to small population sizes and the use of local search in memetic algorithms, the problem of premature convergence arises. Therefore, the restart technique proposed by Eshelman [4] is employed. During the run, it is checked whether the search has converged. If so, the whole population is mutated except for the best individual. The mutation used here exchanges k edges with k being high compared to the mutation operator described above.

3 EXPERIMENTAL RESULTS

Several experiments have been conducted to evaluate the performance of MAs for the TSP. All experiments described in the following were conducted on a PC with Pentium III Processor (500 MHz) under Linux 2.2. All algorithms were implemented in *C++*. For details of the algorithms see [17].

3.1 Comparison of Recombination Operators

In a first set of experiments, several recombination operators for the TSP were tested under the same conditions on three selected TSP instances contained in TSPLIB: att532, pr1002, and fl1577. To get a clear picture of the operator effectiveness, no additional mutation was performed and the restart mechanism was disabled during the runs. Furthermore, a fast *2-opt* local search was used in the MAs that is not as effective as *3-opt* local search or the Lin-Kernighan heuristic to reduce the strong influence of the (sophisticated) local search. The recombination operators MPX, DPX, and the generic greedy recombination operator were studied with various parameter settings. The population was set to $P = 100$ in all runs, and the variation operator application rate was set to 0.5, i.e., 50 offspring were generated per generation. The results of the experiments are summarized in Table 1. For each instance/operator, the average number of generations, the shortest tour length found, and the percentage excess over the optimum solution value is provided. For the GX operator, the values for *cRate*, *nRate* and *iRate* are provided in the form *cRate/nRate/iRate*. For example, a parameter setting of 1/0.25/0.75 means that the common inheritance rate *cRate* was set to 1.0, the new edges insertion rate *nRate* was set to 0.25, and the inheritance rate *iRate* was set to 0.75. The dot in each column block indicates the best result within this block.

For all three instances, MPX and DPX are outperformed by GX for some of the parameter settings: all GX variants with a common inheritance rate of 1.0 and a new edge introduction rate of 0.25 perform better than MPX and DPX. However, the best parameter setting for GX is for each of the instances a different one implying that there is no "golden rule" leading to the best recombination strategy for all TSP instances! For example, the best setting for fl1577 is 1/0/0.75 but all other combinations with *nRate* set to 0.0 do not perform as well as the GX variants with *nRate* set to 0.25. Furthermore, it becomes apparent that respectfulness is a very important property of recombination operators since all GX versions with a common inheritance rate less than 1 perform significantly worse than the respectful greedy recombination operators. However, choosing a high inheritance rate can compensate the phenomenon to an extent since the common edges of the parents have a chance to be included in the offspring in the third phase of the generic recombination. Additionally, iterated *2-opt* local search (ILS) and a MA with the non-sequential four-change mutation (NS4) and no recombination has been applied to the three instances. The mutation based al-

Table 1: Comparison of MA Recombination Strategies for the TSP (2-opt)

Operator	att532		pr1002		fl1577	
DPX	1565	27793.0 - 0.386%	664	266240.5 - 2.778%	653	22314.0 - 0.292%
MPX	2691	27772.0 - 0.311%	3404	261695.5 - 1.023%	1240	22347.8 - 0.444%
GX-Params						
1/1/1	650	27738.7 - 0.190%	307	268183.5 - 3.528%	554	22295.6 - 0.210%
1/1/0.75	708	27744.7 - 0.212%	354	268072.9 - 3.485%	592	22306.7 - 0.259%
1/1/0.5	725	27740.0 - 0.195%	415	267033.1 - 3.084%	585	22304.0 - 0.247%
1/1/0.25	669	27772.0 - 0.311%	304	268487.4 - 3.645%	580	22296.5 - 0.213%
1/0.5/1	868	27729.8 - 0.158%	759	260907.8 - 0.719%	624	22294.8 - 0.206%
1/0.5/0.75	929	27727.0 - 0.148%	733	261981.0 - 1.133%	713	22294.6 - 0.205%
1/0.5/0.5	923	27725.2 - 0.142%	808	261121.2 - 0.801%	682	22296.7 - 0.214%
1/0.5/0.25	892	27723.9 - 0.137%	832	260723.4 - 0.648%	641	22303.5 - 0.245%
1/0.25/0	928	27724.5 - 0.139%	1223	260671.2 - 0.628%	690	22304.5 - 0.250%
1/0.25/0.75	1091	• 27719.2 - 0.120%	1430	260683.9 - 0.633%	769	22294.8 - 0.206%
1/0.25/0.5	1065	27722.4 - 0.131%	1422	260585.9 - 0.595%	684	22311.7 - 0.282%
1/0.25/0.25	998	27723.3 - 0.135%	1334	• 260508.6 - 0.565%	696	22307.0 - 0.261%
1/0/1	956	27763.5 - 0.280%	1321	261379.9 - 0.901%	736	22323.4 - 0.335%
1/0/0.75	1071	27728.0 - 0.152%	1481	260894.8 - 0.714%	735	• 22287.8 - 0.174%
1/0/0.5	1035	27725.4 - 0.142%	1434	260949.5 - 0.735%	744	22312.0 - 0.283%
1/0/0.25	1006	27737.7 - 0.186%	1412	260984.0 - 0.749%	719	22326.2 - 0.347%
0.75/0.5/1	201	28429.8 - 2.686%	226	269423.5 - 4.007%	212	22725.8 - 2.143%
0.75/0.5/0.75	224	28435.5 - 2.707%	254	269423.5 - 4.007%	230	22725.8 - 2.143%
0.75/0.5/0.5	215	28435.5 - 2.707%	243	269423.5 - 4.007%	225	22725.8 - 2.143%
0.75/0.5/0.25	206	28434.8 - 2.705%	232	269423.5 - 4.007%	219	22725.8 - 2.143%
0.75/0.25/0	233	27986.0 - 1.084%	229	269271.2 - 3.948%	227	22679.0 - 1.932%
0.75/0.25/0.75	269	28230.8 - 1.968%	288	269423.5 - 4.007%	269	22671.2 - 1.897%
0.75/0.25/0.5	254	28063.3 - 1.363%	258	269335.2 - 3.972%	254	22657.9 - 1.838%
0.75/0.25/0.25	243	27976.5 - 1.049%	240	269384.7 - 3.991%	239	22649.5 - 1.800%
0.75/0/1	407	27869.0 - 0.661%	422	263536.0 - 1.734%	270	22583.3 - 1.503%
0.75/0/0.75	517	27771.5 - 0.309%	705	• 261696.8 - 1.024%	620	• 22319.3 - 0.316%
0.75/0/0.5	457	• 27747.2 - 0.221%	558	262236.0 - 1.232%	398	22415.2 - 0.747%
0.75/0/0.25	415	27750.5 - 0.233%	435	262634.5 - 1.386%	298	22492.2 - 1.093%
0.5/0.25/0	156	28394.2 - 2.558%	179	269400.0 - 3.998%	161	22725.8 - 2.143%
0.5/0.25/0.75	191	28433.2 - 2.699%	224	269423.5 - 4.007%	187	22725.8 - 2.143%
0.5/0.25/0.5	172	28414.0 - 2.630%	201	269423.5 - 4.007%	178	22724.8 - 2.139%
0.5/0.25/0.25	162	28373.5 - 2.483%	187	269423.5 - 4.007%	170	22725.8 - 2.143%
0.5/0/1	195	28041.8 - 1.285%	216	266696.7 - 2.954%	174	22693.8 - 1.999%
0.5/0/0.75	403	27870.7 - 0.667%	455	• 263020.8 - 1.535%	363	• 22416.0 - 0.751%
0.5/0/0.5	293	• 27838.5 - 0.551%	316	263258.8 - 1.627%	242	22530.1 - 1.263%
0.5/0/0.25	220	27894.7 - 0.754%	227	265673.8 - 2.559%	192	22628.6 - 1.706%
ILS	61365	27777.7 - 0.331%	126457	260683.6 - 0.633%	150797	22369.2 - 0.540%
NS4	744	27860.2 - 0.629%	1438	261922.0 - 1.111%	1633	22304.0 - 0.247%
Time:		60 sec.		120 sec.		200 sec.

gorithms perform relatively well but can not compete with the greedy recombination MAs. For the instance fl1577, the MA with NS4 performs much better than ILS indicating that for this type of landscape search from multiple points (population-based search) is more promising.

In the second experiment, we replaced the fast *2-opt* local search with the Lin-Kernighan heuristic. The population size was set to 40, the variation operator application rate was set to 0.5, i.e., 20 offspring were generated per generation, and restarts were enabled with a diversification rate of 0.3 ($0.3 \times n$ edges were randomly exchanged with n denoting the number of cities). The results obtained from experiments with MAs using DPX, MPX, respectful GX, non-sequential-four-change mutation (denoted NS4) in comparison to the iterated Lin-Kernighan heuristic (ILK) are displayed in Table 2. For each instance/operator pair, the average number of generations, and the percentage excess over the optimum solution value is provided. For the GX operator, the values for *nRate* and *iRate* are provided in the form *nRate/iRate*. *cRate* was set to 1.0 in all experiments. The dot in each row indicates the best result for an instance.

Table 2: Comparison of MA Recombination Strategies for the TSP (LK)

Operator	att532	rat783	pr1002	fl1577	pr2392	pcb3038
DPX	0.030 %	0.004 %	0.023 %	• 0.028 %	0.068 %	0.113 %
MPX	• 0.021 %	• 0.001 %	0.169 %	0.142 %	0.054 %	0.128 %
GX 1.0/1.0	0.030 %	0.007 %	0.036 %	0.055 %	0.042 %	0.132 %
GX 1.0/0.75	0.035 %	0.026 %	0.022 %	0.058 %	0.053 %	0.211 %
GX 1.0/0.5	0.040 %	0.008 %	0.011 %	0.045 %	0.050 %	0.171 %
GX 1.0/0.25	0.043 %	0.006 %	0.013 %	0.051 %	0.047 %	0.146 %
GX 0.5/0.5	0.033 %	0.006 %	0.009 %	0.042 %	0.037 %	0.112 %
GX 0.5/0.75	0.031 %	0.007 %	0.031 %	0.048 %	0.055 %	0.175 %
GX 0.5/0.5	0.035 %	0.008 %	0.005 %	0.046 %	0.051 %	0.143 %
GX 0.5/0.25	0.037 %	0.009 %	0.011 %	0.037 %	0.044 %	0.136 %
GX 0.25/0	0.026 %	0.002 %	0.017 %	0.044 %	0.022 %	0.125 %
GX 0.25/0.75	0.038 %	0.012 %	0.003 %	0.041 %	0.031 %	0.151 %
GX 0.25/0.5	0.035 %	0.006 %	0.002 %	0.036 %	0.025 %	0.111 %
GX 0.25/0.25	0.041 %	0.005 %	0.002 %	0.040 %	0.023 %	• 0.111 %
GX 0.0/1.0	0.045 %	0.008 %	0.006 %	0.052 %	• 0.020 %	0.123 %
GX 0.0/0.75	0.036 %	0.003 %	• 0.000 %	0.043 %	0.027 %	0.115 %
GX 0.0/0.5	0.034 %	0.011 %	0.008 %	0.052 %	0.029 %	0.122 %
GX 0.0/0.25	0.037 %	0.004 %	0.002 %	0.050 %	0.035 %	0.123 %
ILK	0.046 %	0.018 %	0.065 %	0.158 %	0.215 %	0.135 %
NS4	0.055 %	0.010 %	0.020 %	0.181 %	0.119 %	0.171 %
Time:	60 sec.	80 sec.	200 sec.	300 sec.	400 sec.	800 sec.

Here, the performance differences of the MAs are in most cases not significant. For the problems att532, rat783, and pr1002 all algorithms perform well with only small differences, except for the MA with MPX recombination in case of pr1002. Surprisingly, this MA performs significantly worse than the other algorithms. For fl1577, the MAs with DPX and GX outperform all other competitors, with the MA using DPX being the best. For pr2392, all recombination based algorithms perform similarly, but the MAs with mutation and ILK perform significantly worse. In case of pcb3038, the largest instance considered, all results lie close together. The MAs with DPX and MPX outperform ILK and the MA with NS4. In the greedy recombination MAs, high differences can be observed. The best results are obtained with a new edge insertion rate of 0.25. The results show no clear tendency, and often the values lie too close together to be significantly different. However, in none of the cases, ILK or the MA with mutation is able to outperform the MA using DPX or the best greedy recombination. The performance differences between mutation and recombination operators have become more apparent using *2-opt* local search. For larger instances, this may be also observed for MAs with the LK heuristic.

3.2 DPX vs. GX Recombination

Using a NS4 mutation application rate of $m = 0.1$, the MAs have been run on a variety of problem instances contained in TSPLIB, to show the robustness and scalability of the memetic approach. In Table 3, the results are shown for five instances up to a problem size of 1002. The population size was set to $P = 40$ in all runs, the recombination application rate was set to 0.5, and the diversification rate to 0.1. Two MAs were run on each instance, the first one with DPX recombination and the second one with GX recombination. In the latter, *cRate* was set to 1.0, *nRate* was set to 0.1 which appears to be a good compromise between 0.25 and 0.0, and *iRate* was set to 0.5. The programs were terminated as soon as they reached an optimum solution. In the table, the average number of generations (gen) and the average running time of the algorithms (t in s) in seconds is provided. In 30 out of 30 runs, the optimum could be found for all instances in less than two minutes. The average running time for rat783 is

Table 3: Average Running Times of two MAs to find the Optimum

Instance	**Op**	**gen**	**quality**	N_{opt}	**t in s**
lin318	DPX	19	42029	30/30	8
	GX	13	0.00%	30/30	8
pcb442	DPX	824	50778	30/30	147
	GX	286	0.00%	30/30	68
att532	DPX	560	27686	30/30	127
	GX	289	0.00%	30/30	106
rat783	DPX	122	8806	30/30	26
	GX	136	0.00%	30/30	35
pr1002	DPX	333	259045	30/30	112
	GX	182	0.00%	30/30	98

much lower than for att532 due to the structure of the fitness landscapes (see [22] for details): In most cases, the MA with greedy recombination appears to be slightly superior to the MA with DPX.

Additional experiments have been performed on TSPLIB instances up to a problem size of 85900. Due to the limited number of pages in this contribution, the results are not displayed here. They can be found in [22].

4 Conclusions

In an extensive study, several recombination operators including a newly proposed generic greedy recombination operator (GX) are compared in a MA framework. The MAs show significant performance differences if a simple *fast 2-opt* local search is employed. For MAs with the sophisticated Lin-Kernighan local search, the results lie much closer together. The study has shown that respectfulness is the most important property of a recombination operator. Furthermore, we have shown that the MA with the newly proposed greedy recombination operator outperforms all its competitors: MAs with DPX or MPX recombination, MAs with non-sequential four change mutation, and iterated local search.

MAs with DPX and GX recombination and mutation have been applied to various instances contained in TSPLIB to show robustness and scalability of the approach. For problems with up to 1000 cities the optimum could be found in all runs in an average time of less than two minutes on a personal computer with 500 MHz.

References

[1] D. Bonachea, E. Ingerman, J. Levy, and S. McPeak, "An Improved Adaptive Multi-Start Approach to Finding Near-Optimal Solutions to the Euclidean TSP," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, (D. Whitley *et al.*, eds.), (Las Vegas, Nevada, USA), pp. 143–150, Morgan Kaufmann, 2000.

[2] D. Corne, M. Dorigo, and F. Glover, eds., *New Ideas in Optimization*. McGraw-Hill, London, 1999.

[3] J. Dzubera and D. Whitley, "Advanced Correlation Analysis of Operators for the Traveling Salesman Problem," in *Parallel Problem Solving from Nature - Proceedings of the third Workshop, PPSN III*, (H.-P. Schwefel and R. Männer, eds.), (Dortmund, Germany), pp. 68–77, Springer-Verlag, Berlin, Germany, 1994.

[4] L. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," in *Foundations of Genetic Algorithms*, (G. J. E. Rawlings, ed.), pp. 265–283, Morgan Kaufmann, 1991.

[5] B. Freisleben and P. Merz, "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, (T. Bäck, H. Kitano, and Z. Michalewicz, eds.), (Piscataway, NJ), pp. 616–621, IEEE Press, 1996.

[6] B. Freisleben and P. Merz, "New Genetic Local Search Operators for the Traveling Salesman Problem," in *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature - PPSN IV*, (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), pp. 890–900, Springer, 1996.

[7] M. Gorges-Schleuter, "Explicit Parallelism of Genetic Algorithms through Population Structures," in *Parallel Problem Solving from Nature*, (H. Schwefel and R. Manner, eds.), pp. 150–159, Springer-Verlag, 1991.

[8] M. Gorges-Schleuter, "Asparagos96 and the Traveling Salesman Problem," in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pp. 171–174, IEEE Press, 1997.

[9] J. J. Grefenstette, "Incooperating Problem Specific Knowledge into Genetic Algorithms," in *Genetic Algorithms and Simulated Annealing*, (L. Davis, ed.), pp. 42–60, Morgan Kaufmann Publishers, 1987.

[10] W. W. Hsu and C.-C. Hsu, "The Spontaneous Evolution Genetic Algorithm for Solving the Traveling Salesman Problem," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, (L. Spector *et al.*, eds.), pp. 359–366, Morgan Kaufmann, 2001.

[11] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study," in *Local Search in Combinatorial Optimization*, (E. H. L. Aarts and J. K. Lenstra, eds.), pp. 215–310, Wiley and Sons, New York, 1997.