Dynamic Operators in Child Creation and Fitness for Improved Performance of a Genetic Algorithm

David C. Rich*
149 E. Countryside Circle
Park City, Utah 84098-6102, USA
Dave_Rich@out.trw.com

ABSTRACT

A genetic algorithm has been applied to optimizing a university class schedule. A complete description of the algorithm is beyond the scope of this paper, which will address the adaptive controls applied to the progress of the Genetic Algorithm. The benefits of dynamic operator selection in the genetic processes of child creation are described. A dynamic penalty function also guides the fitness and population selection of better-fit solutions. These adaptive controls are not inclusive of all possible adaptations but only hint at the improvements that can be achieved by using dynamic controls.

KEY WORDS

dynamic, adaptive controls, Genetic Algorithms, dynamic creation, dynamic penalty function

1 INTRODUCTION

A steady-state, value-based GA is used to explore the large search space of potential university class timetables [5, 6]. The steady-state GA allows the new chromosomes to immediately become part of the population. The child is kept or discarded according to the results of a fitness tournament, with the fitness determined by a penalty function. The value of each gene is an integer representing the start time of the first period that the class meets each week. Dynamic adaptation was used in the birth of new individuals as well as the population evaluation and the ranking and selection of the more fit individuals.

2 DYNAMIC CONTROLS IN CHILD CREATION

The common methods used for creating children in a GA are crossover and mutation. Many researchers have experimented with different types of crossover as well as different mutation rates. Typically, both crossover and mutation are applied to the creation of the same child. In this application, these two operators were separated and applied separately. That is, either crossover or mutation was used. A major objective in separating the two is the analysis of the results. If both operators are applied at the same time, it is difficult to determine which of the two operators resulted in an improvement--when there is one--in the fitness of the resulting child. By separating the two operators and applying them individually, it is possible to track the origin of improvements.

---

* Dave Rich has worked in nuclear physics engineering and research for over twenty years. He works full time for TRW, teaches part time for Utah Valley State College, and does consulting.

The probability of selecting either crossover or mutation is dynamic. The initial probabilities for each one was 50%. Each operator selection rate changes during a run according to the success of each operator in improving the fitness of the best chromosome [2, 7]. As one operator provides more improvement in the fitness, its selection probability increases at the expense of the other operator. As a result, one operator can dominate in the early generation, which crossover often does, while the other operator, normally mutation, can dominate in the later population improvement. Each operator is assigned a base probability that it cannot go below. This maintains the integrity of always having a finite probability and not "losing" an operator. In this study, the base probability was maintained at 20%.

Each operator has a success rate that keeps track of the fraction of the child chromosomes created by it that remain after each fitness competition. Each operator also has an integer credit that is incremented each time that the operator generates a chromosome with a fitness better than the previous best fitness. The "add-on" selection probability for mutation is calculated as the ratio of the mutation "credit for best" to the summation "credit for best" plus the ratio of the mutation "success rate" to the summation "success rate." The total mutation selection probability is the sum of the base selection probability plus the add-on selection probability:

```
sumCredit       = mutate.creditForBest + xover.creditForBest
sumSuccess      = mutate.successRate + xover.sucessRate
mutate.addProb  = mutate.creditForBest / (1. + sumCredit) + mutate.successRate / (1. +
                  sumSuccess)
mutate.opProb   = mutate.baseProb + mutate.addProb
```

The crossover operator selection probability is calculated as one minus the mutation operator selection probability:

```
xover.opProb    = 1. - mutate.opProb
```

The dynamic operator selection probabilities were different in every run. In some runs, there would be large excursions in the selection probabilities, while in others the selection probabilities would be close to 50%. They would vary for each run as a different part of the scheduling landscape was explored. The dynamic selection rate, however, would progress toward the operator with the currently-best success rate. At the beginning of a run, the small numbers of successes for each operator would result in a few large changes before settling down to more gradual changes. Figure 1 displays a sample of the changing operator selection probabilities.

In this run, crossover dominated the selection probabilities for over two hundred chromosome generations. The dominance then changed dramatically to favor mutation, which almost achieved the maximum selection probability of 80%.

The later dominance by mutation is not always the case. Figure 2 displays a different operator selection graph. In the run represented by this graph, crossover dominated for a short time, then mutation took over. However, mutation only dominated for a short time before crossover became dominant, again. As the run continued, both crossover and mutation selection probabilities remained close to 50%.

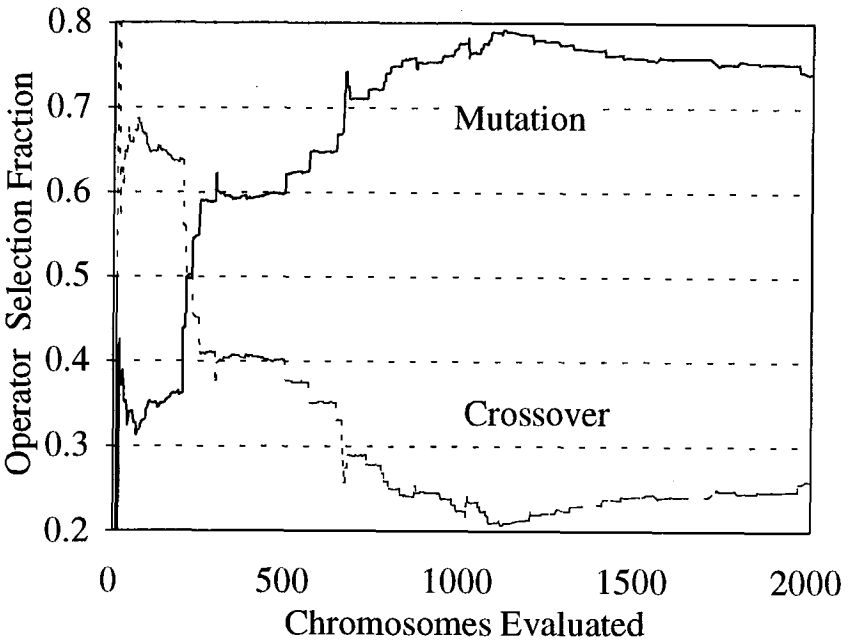

Figure 1. Dynamic operator selection rates of mutation and crossover during the generation of 2000 chromosomes. The selection rates are adjusted according to the success of each operator to create chromosomes that improve the population fitness.

The benefit of dynamic operator selection is that the selection rates can change according to the benefit to the population progression--without the programmer being required to preprogram or have prior knowledge of where these changes should occur. And the changes should occur at different times in the operation, depending on the domain that is randomly selected for searching. Because this random domain selection cannot be known in advance, it is impossible for the programmer to hardwire the best operator selection rates

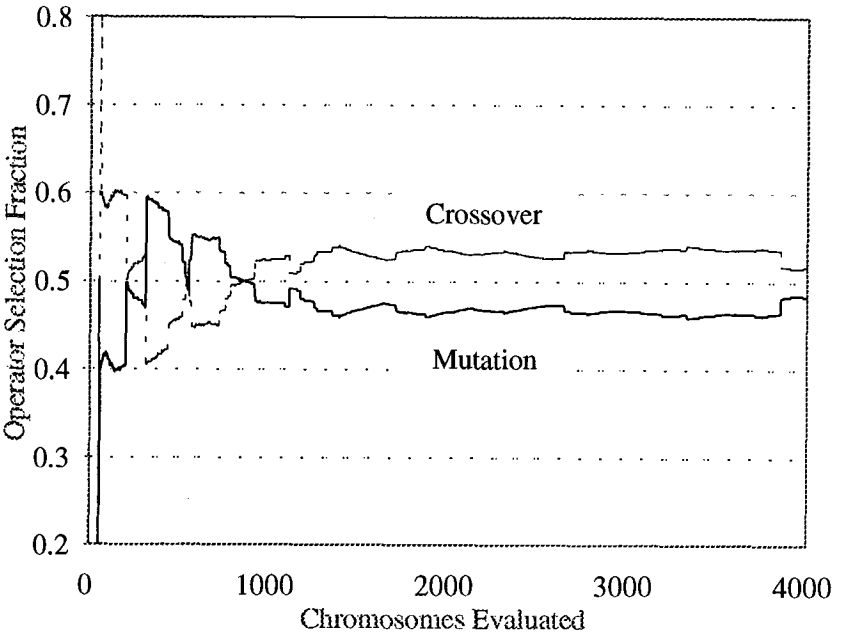

Figure 2. In this run, crossover dominated, then mutation took over, and then crossover dominated, again. The operator selection probabilities of both operators remained close to 50%.

for every potential landscape. The important concept is that whatever operators are selected, the population progression will be enhanced by creating a dynamic selection process for the operator set. This will allow the population to explore any area of the search landscape with prime operator selection. When small changes need to be made, mutation can be emphasized, with a potential to try large changes through crossover to jump to a new part of the landscape. When large changes are needed, crossover can be emphasized.

3 DYNAMIC PENALTY FUNCTION

The fitness of a chromosome is calculated, in this study, using a penalty function. A penalty function allows the opportunity to gain information from data points that are valid for the GA but not for the problem. Instead of discarding genes that do not satisfy the physical constraints, they are assessed a cost that causes the GA to move toward more acceptable results. This study addresses both hard constraints and soft constraints. Hard constraints are those constraints that can never be violated in an acceptable schedule. Instead of disallowing these data points during the processing, these points are assessed a high cost. All hard constraints have the same high cost, since a schedule is unacceptable if any hard constraint is violated. The high cost does not require an immediate repair of an unacceptable solution but causes evolutionary pressure to move the solution away from the hard constraints.

Soft constraints are those things that would be nice to have (or not have) but their lack (or existence) does not make a solution unacceptable. Soft constraints are restrictions that can be allowed but are not preferred. These are assigned a much lower cost than hard constraints, so that the hard constraints can take precedence over the soft constraints during evolutionary search procedures. The costs of the soft constraints can also be varied according to the relative value of each constraint.

The hard constraint costs are added to the soft constraint costs to calculate the chromosome fitness. The fitness of the chromosome is the inverse of the cost, which results in a value (fitness) for each chromosome that is in direct proportion to its value as a solution:

$$\text{fitness} = 1. / (1. + \text{cost})$$

The fitnesses of all chromosomes in the population are summed, and a fitness ratio is assigned to each chromosome. The fitness ratio is the chromosome fitness divided by the summation of all fitnesses, which identifies the value of each chromosome relative to the other chromosomes. The summation of all fitness ratios is 1.

As the population of schedules evolves toward better and better schedules (chromosomes), there will come a point where the schedules are all equally good, or almost so. This point of almost-equal fitness values is called convergence. Beasley [1] defines convergence as, " . . . progression toward increasing uniformity." In this project, when the average chromosome fitness is within 95% of the best-fit chromosome, the population is said to have converged. After convergence, there is very little improvement that can evolve through crossover, since the chromosomes are all almost alike. Figure 1 implies that crossover is only effective for a few hundred chromosome generations.

Since the operator selection rate is based on the success of each operator, the program could be left to continue to exploit mutation at an increasing rate. However, greater benefits can be achieved by having a break in the procedures after convergence.

When the population converges, all but the best-fit chromosome are reinitialized to random values within the legal gene range. Reinitializing a converged population provides for greater diversity and improves the search process [3]. The best fit chromosome is not reinitialized but is kept, which is called elitism. Elitism provides some assurance that vital information already discovered is not lost through the reinitialization process. After reinitialization, the search continues with the new chromosomes blending their genes with those from the best chromosome previously discovered. This break in the search process not only expands the domain searched, but also allows for an operational point where the penalty function can be made dynamic.

Constraint violation costs are made dynamic at the point of population reinitialization. The scheduler begins with each constraint associated with a basic cost and a basic cost increment. Each time the population converges and is reinitialized, the best-yet chromosome will (probably) have some constraints unresolved. The cost associated with each constraint violated by the best-yet chromosome is incremented by the associated cost increment. The increasing constraint costs put pressure on the scheduler to increase the priority in resolving those constraints. The greater the cost of an individual constraint, the greater is the pressure for that constraint to be resolved in the next evaluation. The slowly increasing costs on violated constraints create increasing pressure on the scheduler to find the lowest-energy, most stable solution.

With the total costs dynamic, the important factor is the relative cost increment. Those constraints with the larger increments receive greater pressure more rapidly and will be resolved more quickly. As violated constraint costs are increased, those constraints will also be resolved until the best possible schedule is discovered. In the real world, the physical resources (rooms) rarely match exactly with the required resources. Typically, the physical resources are greater than the minimum required, and there may be multiple timetable configurations that have a minimum-energy, or almost-minimum-energy, class schedule.

Constraints that are difficult to resolve can generate very large costs through multiple convergence cycles. Sometimes, the cost of a constraint can be increased significantly but then become dormant and insignificant when the constraint is satisfied. If the constraints would stay satisfied, the large dormant costs would be irrelevant. However, as other constraints are satisfied, some previously-satisfied constraints may become violated again due to the highly interrelated, or epistatic, environment. The constraints forced into renewed violation when some other constraint is resolved, are immediately assessed the large dormant cost previously built up.

It is possible to avoid some of these down-the-road surprises by starting over. This program is designed to accept a seed chromosome as a member of the initial population. Therefore, the scheduling process can be run for a specified number of chromosome generations and then stopped. The best chromosome is then entered as a seed to a new run begun with all costs initialized to the basic costs. As a result, the scheduler begins working with a best-yet chromosome but at the beginning of the dynamic cost cycle.

Dynamic costs make it difficult to compare the best chromosome in one cycle with the best chromosome in another cycle after some constraint costs have increased. To overcome this difficulty, a "normalized cost" of the best-yet chromosome is calculated. The normalized costs used are the cost increments that have been entered relative to the importance of each constraint type. These increments do not change. The summation of cost increments of all violated constraints therefore provides a good comparative (normalized) measure of each chromosome's value throughout the many cost-increasing convergence cycles.

The normalized costs can also increase as well as decrease. When one constraint is increased in priority with the increasing penalty costs, it may dislodge other less-penalizing constraints. This can actually result in an increase in the normalized cost even when the total cost decreases. A solution to this problem has not yet been designed, except to continue with the evolution.

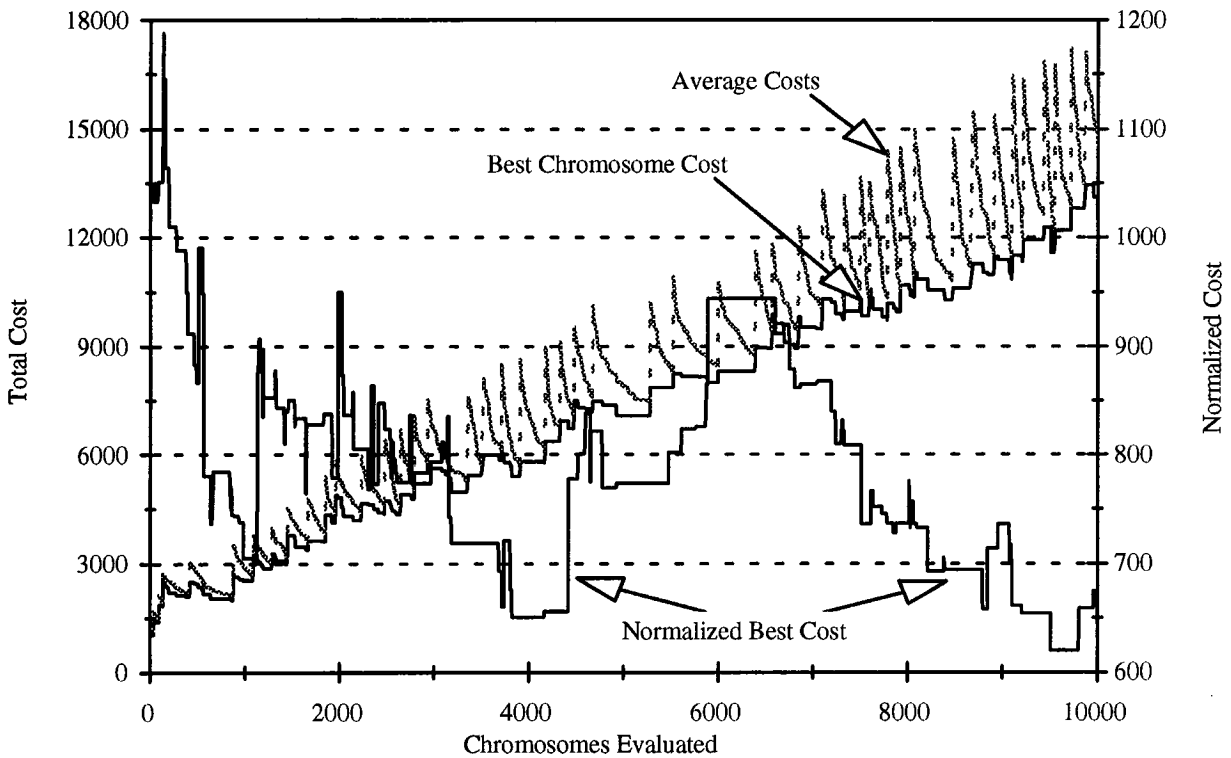

Figure 3. Cost data showing the increasing costs using a dynamic penalty function and the normalized cost of the best-fit chromosome. This run was extended to 10,000 chromosome evaluations to allow the dynamic costs more opportunity to force improvements in the solution. The best schedule found still had a cost of 620, with many hard constraints violated. The acceptable solution had to be found using the dynamic penalty function in conjunction with other controls.

Figure 3 shows the costs of a sample run using a dynamic penalty function. Note that at about 4000 evaluations, the normalized cost of the best chromosome began to increase. However, later in the run, the normalized cost was forced back down lower than it had been before the increase.

4 SUMMARY

This study has shown that dynamic controls applied to Genetic Algorithms helps the GA search the potential landscape more efficiently from emphasizing large changes to emphasizing small changes. They allow the program to modify the search procedures to the best advantage of the local position in the search space, thereby improving the speed of the search. The methods applied herein are only examples. Recognizing the potential for improved performance, continued research will undoubtedly discover more and better dynamic controls to enhance evolutionary performance.

REFERENCES

1. Beasley, D, Bull, D.R., and Martin, R.R. An Overview of Genetic Algorithms: Part 1, Fundamentals. In *University Computing*. 15, 2 (1993), 58-69.

2. Davis, L. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

3. Karr, C.L. Air-Injected Hydrocyclone Optimization via Genetic Algorithm. In *Handbook of Genetic Algorithms*, by L. Davis, 222-236. Van Nostrand Reinhold, New York, 1991.

4. Khuri, S., Bäck, T., and Heitkotter, J. An Evolutionary Approach to Combinatorial Optimization Problems. In *Proceedings of the Computer Science Conference, 1994.* March 8-10,1994. ACM Press.

5. Rich, D.C., Automated Scheduling Using Genetic Algorithms, *2nd Annual Utah Workshop On: Applications of Intelligent and Adaptive Systems,* The University of Utah Cognitive Science Industrial Advisory Board and The Joint Services Software Technology Conference '94, April, 1994.

6. Rich, D.C., A Smart Genetic Algorithm for University Timetabling, *1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT),* Napier University, Edinburgh, Scotland, UK, August, 1995.

7. Tate, D.M. and Smith, A.E. Dynamic Penalty Methods for Highly Constrained Genetic Optimization. Submitted to *ORSA Journal on Computing*, (Aug. 1993).