The G*ENITOR* Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best

Darrell Whitley
Computer Science Department
Colorado State University
Fort Collins, CO 80523.
whitley@cs.colostate.edu

Abstract

This paper reports work done over the past three years using rank-based allocation of reproductive trials. New evidence and arguments are presented which suggest that allocating reproductive trials according to rank is superior to fitness proportionate reproduction. Ranking can not only be used to slow search speed, but also to increase search speed when appropriate. Furthermore, the use of ranking provides a degree of control over selective pressure that is not possible with fitness proportionate reproduction. The use of rank-based allocation of reproductive trials is discussed in the context of 1) Holland's schema theorem, 2) DeJong's standard test suite, and 3) a set of neural net optimization problems that are larger than the problems in the standard test suite. The G*ENITOR* algorithm is also discussed; this algorithm is specifically designed to allocate reproductive trials according to rank.

Background and Motivation

In 1985 James Baker reported experiments where reproductive trials were allocated according to the rank of the individual strings in the population rather than by individual fitness relative to the population average. His results were fairly successful, but seem not to have revolutionized the way people think about genetic algorithms. There are several reasons why his results have not received as much attention as they perhaps deserve. First, Baker used ranking to slow down convergence: this not only resulted in more accurate optimization, but also slower optimization. The experiments reported here indicate that it is also possible to use ranking to achieve faster search on easier optimization problems *and on more difficult optimizations problems, where a reduced but constant selective pressure can yield steady improvement resulting in more efficient optimization.*

A second reason that ranking has not received more attention may be the standard test suite itself. The test problems are all rather small and relatively easy. It is quite common for researchers to try to solve these problems using as few recombinations as possible–or to compare optimization after some fixed number of recombinations. These measures can be deceptive. The G*ENITOR* algorithm shows dramatic increases in performance on larger problems such as the neural net experiments reported here.

A third reason that ranking has probably not received more attention is that it does not at first appear to be consistent with Holland's fundamental theorem of genetic algorithms: the schema theorem. The schema theorem predicts changes in the sampling rate for a hyperplane from generation t to generation $t+1$. This change satisfies the inequality

$$P(h,t+1) \geq \left\{1 - P_c \frac{l(h)}{L-1}\,(1-P(h,t))\right\} FR\; P(h,t)$$

where h represents a particular hyperplane and $P(h,t)$ indicates the fraction of the population that are members of a hyperplane h at time t. FR indicates the average fitness of the members of h relative to the population average (the "Fitness Ratio" of hyperplane h.) The defining length of the schema, $l(h)$, includes only the significant part of the defining schema for h. L is the length of the binary string or "genotype." P_c is the probability that crossover will be applied (Holland 1975).

The schema theorem calculates the number of representations that a particular schema has in the population during one generation and calculates the average fitness of the individuals that possess the schema in question; this value, which will be referred to as S_f, is compared to the average fitness of the population, P_f, to determine the fitness ratio (ie: $FR = S_f/P_f$).

One of the beauties of genetic search is that there is no need to ever directly manipulate schemata. In actual implementation the number of trials that any individual receives during any generation is the individual fitness I_f divided by the average fitness of all the individuals in the population P_f. Thus I_f is actually a partial evaluation of the $2^L - 1$ schemata that make up an individual genotype.

The problem with ranking is that it doesn't seem to fit with the schema theorem–but in fact there is no reason that FR needs to be calculated as a direct function of I_f. Instead of using $I_f/Psubf$ to calculate the fitness ratio, $FR = Rank(I_f)$ can be substituted without changing the intent of the schema theorem. In some ways it is actually *more* consistent with the schema theorem because it removes the need for other "parameters" that are used to indirectly control selective pressure that are not part of the schema theorem.

This raises another issue. It can be argued that there are only two primary factors (and perhaps only two factors) in genetic search: population diversity and selective pressure. These two factors are inversely related. Increasing selective pressure results in a faster loss of population diversity. Maintaining population diversity offsets the effect of increasing selective pressure. In some sense this is just another variation on the idea of exploration versus exploitation that has been discussed by Holland and others. Many of the various parameters that are used to "tune" genetic search are really indirect means of affecting selective pressure and population diversity. As selective pressure is increased, the search focuses on the top individuals in the population, but because of this "exploitation" genetic diversity is lost. Reducing the selective pressure (or using a larger population) increases "exploration" because more genotypes and thus more schemata are involved in the search.

The research reported in this paper is based on the idea that selective pressure and population diversity should be controlled as directly as possible. The better one can understand and control the relationship of population diversity and selective pressure to the parameters used to "tune" a genetic algorithm, the more insight the researcher will gain toward improving genetic search. Selective pressure can be simply and directly controlled by allocating reproductive trials according to rank.

Scaling Problems and Ranking

Fitness proportionate reproduction can sometimes lead to problems when conditions arise where the search is likely to (1) stagnate because the search lacks selective pressure, or (2) prematurely converge because selection has caused the search to narrow too quickly. The most usual cause of this is referred to as the "scaling problem." Suppose the genetic algorithm is applied to a maximization problem where the performance values of genotypes in the population range from 100 to 1100 with an average of 550 and the fitness ratio is calculated using individual fitness over the population average. Initially, the selective pressure toward the top ranked genotype is 1100/550 or 2. Suppose this is a sufficient selective pressure to keep the search moving forward but so much as to cause premature convergence. However, later in the search the range may be 1000 to 1200 with an average of 1100. Now the selective pressure is only 1200/1100 or 1.09, which may not be adequate and the search stagnates. (These values should not be taken as being precise definitions of "appropriate selective pressure;" they are simply used to indicate that selective pressure can fluctuate.) To fix this, we could subtract 1000 from all the values in the above example, creating an effective range of 0 to 200, an average of 100, and a selective pressure of 2.0 again toward the top ranked genotype. Maintaining adequate selective pressure as a population becomes more homogeneous is difficult because there is less variation in fitness. John Grefenstette's *Genesis* implementation package and user's guide (Grefenstette 1984) defines various "scaling window" options that work in this general fashion. But this represents a deviation from a strict interpretation of the fundamental theorem of genetic algorithms. Ranking is in some ways more consistent with the schema theorem because it makes it unnecessary to introduce additional parameters that are not accounted for by the schema theorem in order to control selective pressure. Selective pressure remains consistent throughout the search; scaling problems do not occur and fitness differences between genotypes can be exploited, regardless of the magnitude of those differences.

Baker used ranking in an effort to stop premature convergence. One cause of premature convergence may be "super genotypes" that have an unusually high fitness ratio and thus dominate the search process; Baker seems largely concerned with slowing down searches that progress too fast because of "super individuals." But ranking completely solves the scaling problem. Consider the following "lock and tumbler" problems (Ackley 1987; Davis 1988, personal communication). Starting at the first bit in a genotype, let C be the number of "1's" that occur in some predefined sequence. For example, if the sequence is 0,1,2,...L, then C would be the number of consecutive 1's that occurs as a prefix of the binary encoding. Thus, the string 1111101011...1 has $C = 5$. Notice that 1's which are not part of the initial prefix do not contribute to the value of C, only those that occur before the first 0 in the sequence. (The "lock" sequence need not be 0,1,2,...L, but any sequence; e.g. 5,20,1,...etc.) The C representing the bit count could be directly used as an evaluation function, but suppose this did not produce an adequate selection pressure. Another variation on this would be to use an evaluation function where the "evaluation" is C*10, or 50 in this example. Yet another possibility is to let the evaluation function be C! or 5! in this example. In this case "super individuals" would be an enormous problem because of the relative difference in value between individuals with different evaluations. For a genetic algorithm using fitness proportionate reproduction these evaluation functions create very different scaling problems. The task we are trying to solve, however, is really the same in every case. To a rank-based algorithm such as *GENITOR these problems are all identical* and there would be no difference in performance no matter which evaluation function is used.

The use of various "scaling windows" is just one of several parameters aimed at controlling selective pressure or population diversity. Grefenstette (1986) discusses the following parameters: 1) population size, 2) crossover rate, 3) mutation rate, 4) generation gap, 5) scaling window, and 6) selection strategy. All of these in some way affect the selective pressure. In developing the *GENITOR* algorithm, there has been an explicit effort to remove as many parameters from the algorithm as possible. The *GENITOR* algorithm allows population diversity and selective pressure to be directly controlled.

Ranking, Fitness and Schema Theory

The most serious objection to ranking is that it violates the schema theorem. It might be argued that the average of the rank of the genotypes that sample a particular hyperplane does not correspond to the rank of the hyperplane's average fitness. But looking at ranking in this way fails to reveal what ranking actually does in the search space. Ranking acts as a function transformation that assigns a new fitness value to a genotype based on its performance relative to other genotypes; in other words, rank translates into a assigned fitness value. Further, it assigns fitness values so as to consistently "distance" the corners in the hypercube that are currently being sampled in terms of their relative fitness–thus acting as a kind of smoothing function. Notice that because of the smoothing effect that ranking has, the hyperplane with the "best average raw fitness" may not be the same as the "best average assigned fitness," but this is exactly the kind of effect one wishes to achieve with ranking. It reduces the effect of exagerated differences in fitness.

If "disruptions" are ignored (and disruptions are the same whether one uses ranking or fitness proprotionate reproduction) the bottom line is this: ranking will increase the representation of schemata that have above average (mean) "assigned fitness" and reduce representation of schemata that have below average "assigned fitness." The implicit parallelism is the same; the actual fitness values and thus the resulting representation of schemata will be different, but information about $2^L - 1$ schemata is still gained with every string evaluation. When the ranking function is linear, the mean "assigned" fitness will correspond to the median rank in the population . When a nonlinear function is used, the mean fitness will be shifted toward the top of the population. It is true that if a nonlinear function is used to assign fitness the genetic algorithm can display hill-climbing tendencies. However, controlling the selective pressure controls the behavior of the algorithm.

A second objection is that ranking discards or ignores information about the search space as revealed by the evaluation function. This objection again fails to see ranking as a function transformation that *intentionally* discards information about the magnitude of fitness and uses relative magnitude instead. In many real applications the evaluation function is likely to be a heuristic that simply indicates which strings are better than others. In most cases it may not be realistic to use the value generated by the evaluation function to judge relative differences in fitness. Ranking (or even an approximate ranking) may the best one can expect from an evaluation function. There is simply no reason to believe that most evaluation functions yield an "exact" measure of fitness. And if they do not yield an "exact" measure of fitness, then why insist that "exact fitness" be used?

The G*ENITOR* Project

G*ENITOR* is an acronym for G*EN*etic *I*mplemen*TOR*, a genetic search algorithm that differs in two major ways from the standard genetic algorithms. One major difference is the explicit use of ranking. The second difference is that G*ENITOR* abandons the generational approach and reproduces new genotypes on an individual basis. It does so in such a way that parents and offspring can and typically do co-exist. Theoretical analyses suggests that G*ENITOR* may be less biased against schemata with long defining lengths than the standard genetic algorithm. Because G*ENITOR* uses a one-at-time reproduction scheme, it also appears to achieve faster feedback relative to the rate at which new points of the search space are being sampled.

G*ENITOR*'s ability to sample high performance hyperplanes is captured in the following formula:

$$P(h,t+1) \geq P(h,t)\ (P(h,t) - \frac{P(h,t)}{P_z\ FR})$$

$$+ \frac{P(h,t)}{P_z}\ FR \left[1 - (1 - P(h,t)\ FR)\ \frac{1}{2}\left\{1 + \frac{l(h)}{L-1}\right\}\right]$$

where P_z is the population size and $\frac{P(h,t)}{(P_z\ FR)}$ is used to estimate the probability that some member of h is replaced. FR is the fitness ratio. Note that both parents are selected using FR to bias selection toward high performance genotypes. However, the theorem is independent of how that fitness ratio is derived–it can either be calculated in the usual way or calculated as a function of the genotype's rank in the population. A derivation and in-depth discussion of this formula is presented in (Whitley 1988b).

G*ENITOR* only produces one new genotype at a time, so inserting a single new individual is relatively simple. Furthermore the insertion automatically ranks the individual relative to the existing pool–no further measure of relative fitness is needed.

Duplicates and Selective Pressure

One way to increase a genetic algorithm's ability to accumulate high performance schemata is to apply crossover probabilistically, which is one idea behind the P_c (the crossover rate) in the standard schema theorem. In G*ENITOR* this could be done by making a copy of the parent and introducing it as the new offspring. If P_c is introduced into the G*ENITOR* formula, it does in fact theoretically increase the algorithm's ability to retain information about long schemata. However, this approach supplies no new information to the search since it does not sample a new genotype and its corresponding schemata.

Probabilistic crossover (P_c) allows some genotypes to remain intact. This allocates more trials to high performance genotypes since they are the most likely to produce duplicates during the reproduction/selection phrase. The result is more copies of what appear to be high performance schemata, but this process fails to test these schemata (i.e. hyperplanes) in a new context until these duplicates undergo recombination. But while this creates additional selective pressure toward schemata contained in the duplicate genotype, it does so in the worst possible way: every duplicate means a loss of genetic diversity in the population. The G*ENITOR* algorithm requires no such safeguard to guarantee the survival of the better genotypes in the population.

The genetic algorithm community often cites the need to balance exploitation and exploration. However, this highlights a deeper conflict in genetic search–population diversity versus selective pressure. Increasing the selective pressure tends to reduce diversity and increase search speed. Decreasing the selective pressure helps to maintain diversity, resulting in a slower, although more robust search. This observation is not novel to biological researchers. However, this observation does add legitimacy to our emphasis on selective pressure and population size as *the* critical variables in a genetic algorithm. Fitness proportionate reproduction does not automatically translate into "appropriate" selective pressure. In fact, as was the case with the "lock and tumbler" described in Section 2, the "exact" value returned by the evaluation function can vary depending on how the evaluation function is implemented. As another example, one can always square the value returned by the evaluation function. For feed-forward neural networks the error at the individual nodes is typically squared and then summed to generate the error term. Squaring error at output nodes can even cause genotypes to be ranked differently: e.g. $1 + 5 < 3 + 4$, but $1 + 25 > 9 + 16$. The point is that the notion of an "exact" fitness value is more arbitrary than it would at first appear. Calculating fitness as a function of rank is a simple and

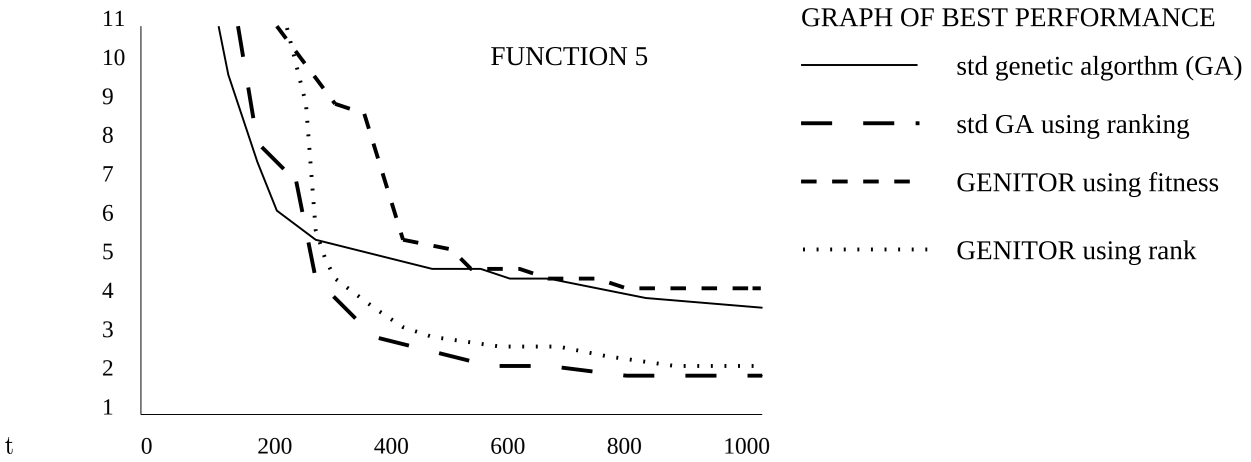

Figure 1: This graph shows best performance on function f5 from DeJong's standard test suite. The tests were averaged over 100 experiments. The horizonal axis indicates the total number of genotypes evaluated: 100 in the initial population, plus 900 generated by crossover. Performance is given on the vertical axis.

effective way to obtain a greater degree of control over selective pressure.

In the empirical tests reported in this paper the "reduced surrogate" 2-point recombination operator developed by Booker (1987) was used. The importance of this operator is stressed here for two reasons. First, it can actually *reduce* the bias toward high ranking genotypes by preventing duplicates; this will help prevent them from prematurely dominating the search. This is also one of the goals of G*ENITOR*; by removing P_c as a variable there will be fewer duplicates and more information about hyperplanes in the population. This will not only reduce premature convergence, it will also mean that more schemata will be tested in a larger number of contexts. Selective pressure is most fairly distributed in the population when there are no duplicates. For example, duplicates that are ranked 5th and 6th in the population can end up with more selective advantage than the top ranked individual in the population. Preventing duplicates removes an unwanted source of selective bias in the G*ENITOR* algorithm.

Second, there is a subtle heuristic at work in the operator which we had been working on independently from Booker. By looking at the reduced surrogates, one is also heuristically reducing the search space. In other words, the two parents are in agreement about the values of certain bit positions. This agreement has the affect of reducing that part of the search space which is currently of interest, especially if one looks at the amount of agreement that exists across several recombinations. The reduced surrogate operator reflects that agreement. This insures that offspring are selected from a part of the search space about which the two parents are not in agreement. In particular offspring generated in this way lie along a minimal edge path between the two parents in a hypercube corresponding to the problem encoding. Consider the following two strings: 0010000000 and 0000010100. By removing all bits that match between the two strings we obtain –1–0-0– and –0–1-1–. Given that one crossover occurs between these "end bits," the offspring will always be different from the parent reduced surrogates. Crossing the reduced surrogates also means that a select part of the search space has been focused on. Over numerous recombinations, this will focus more search toward those parts of the search space that are least agreed upon in terms of their representation in the population.

Empirical Results: The Standard Test Suite

The G*ENITOR* program has been run on five functions that are used as standard test functions for genetic algorithms.

The standard genetic algorithm package used for comparative purposes is the *Genesis* program (developed and graciously provided by John Grefenstette of the Naval Research Laboratory.) In the original implementation of G*ENITOR* the program checked for and eliminated duplicates; *that is not necessary in the current implementation.*

We found duplicates could be controlled by reducing the bias or selective pressure (i.e. by regulating the number of reproductive trials allocated to the top ranked genotypes). The reduced surrogate crossover operator also helped. Again, both algorithms used Booker's 2-point reduced surrogate crossover.

Typically researchers use four standard ways to measure performance (DeJong 1975): online, offline, average, and best. Results for all 4 measures and function descriptions have been presented elsewhere (Whitley 1988a). Also, since G*ENITOR* varies in two ways from a standard genetic algorithm, to accurately evaluate the difference between the two algorithms all four variations were tested: 1) generational reproduction using fitness *(Genesis)*, 2) generational reproduction using ranking, 3) one-at-a-time reproduction using fitness, and 4) one-at-a-time reproduction using ranking (G*ENITOR*). One thing the results demonstrated is that sometimes search can be improved by controlling the selection bias. This is evident in those cases where the two ranked approaches did much better than the fitness proportional approaches. An example of this is given in Figure 1 for function 5 (f5); function 5 tests the algorithms' ability to achieve a global search. Considering all five functions, G*ENITOR* performed best on f1, f3 and f4. No clear winner emerged on f2. On f5, as shown in Figure

1, the rank based generational approach did as well as (or slightly better than) *GENITOR*. *GENITOR* competed well by all measures; the standard genetic algorithm never did much better than *GENITOR*, but *GENITOR* often did much better than the standard genetic algorithm.

Empirical Results: Three Neural Networks

The standard genetic algorithm has been tuned on these same test functions for over a decade (DeJong 1975). The optimization problems posed by neural networks provide a new and more challenging test of genetic algorithms. The genetic algorithms were used to optimize the connection weights in three neural problems. These are 1) the exclusive-or (Xor) problem, 2) a 424-encoder, and 3) an addition problem (Rumelhart 1986). The following discussion assumes some familiarity with back propagation and feed-forward neural networks (Whitley 1988c).

To optimize neural net connection weights, the problem must first be encoded as a binary string. It is assumed that (1) nodes and arc connections are already known to exist (i.e. the net is already configured) and (2) each arc weight can take on a predefined finite range of settings. The genetic algorithm generates a binary string representing arc weights in the net. The network is then run in a feed-forward fashion for each training pattern just as if one were going to use back propagation. The sum of the squared error in then accumulated; this value represents the "performance" of the binary genotype.

Eight bits were typically used to represent each connection weight; on the 424 problem 4 bits were used. These eight bits were interpreted as a signed magnitude number ranging between -127 and +127 with 0 occurring twice (-0 and +0).

The Xor problem appears to be similar in difficulty and size to the standard test problems. It is encoded using 72 bits and can be solved after 500 to 1500 recombinations. Both *GENITOR* and *Genesis* solved the problem easily. Using a population of 200 and using 1500 recombinations *the average error was 0.00001 per run using* *GENITOR*. The average error for *Genesis* under the same conditions was 0.006. A t-test comparing the results for online and best performance indicates these were significantly different at the .01 level. Offline was not significantly different, largely because *Genesis* had better early performance, but *GENITOR* had much better later performance. On very small populations (i.e. a population size of 50) *GENITOR* did not perform as well as *Genesis.*

On the other hand, the performance of *GENITOR* improved as population size was increased. Population size did not have as much effect on the performance of *Genesis.*

The 424-encoder was implemented using 4 bits per link; the total length of the encoding was 88 bits, but the solution space was much more difficult to search. Initial results were averaged over 50 runs using a population of 200. A t-test indicates that offline, online and best performance are all significantly different at the .01 level. Here *GENITOR* did noticeably better, but the total average error is not really satisfactory. However, it was found that better solutions could be generated using larger population sizes for *GENITOR*. An average error of 0.1 was achieved by increasing population size to 1000, but to obtain these results required 50,000 recombinations. Increasing the population with *Genesis* did not seem to help, at least not within a reasonable number of recombinations.

The adder problem involves adding 2 2-bit numbers. This version of the adder problem had 2 2-bit inputs, 4 hidden nodes and 3 output nodes. The layers were fully connected and encoded using 280 bits, 8 bits per connection. For this problem results are averaged over fewer runs because of the long execution times involved. Initial results compared *GENITOR* and *Genesis* on the adder problem with a population size of 2000 after 100,000 recombinations. Here *GENITOR* did notically better than *Genesis,* but only reduced error to 2.48 (averaged over 5 runs). *Genesis* had only reduced error to 5.8 after 100,000 recombinations (averaged over 13 runs). The t-test indicates that both the offline and best performance are significantly different at the .01 level. Online performance was not significantly different. The *Genesis* code was run again and allowed to continue until it achieved similar average best performance (2.48) with a population size of 2000; It required 1,250,000 recombinations to reduce average error to 2.48–more than 5 times the number of recombinations required by *GENITOR* (averaged in this case over 8 runs). With a population of 5000 the *GENITOR* program reduced the average error to 0.5 after 500,000 recombinations (averaged over 5 runs). Larger populations have not been attempted with *Genesis* because of the long execution time required.

These are not excellent results. We have subsequently achieved much better neural net optimization using increased selective pressure coupled with a special mutation operator to sustain diversity. This combination resulted in optimization with greater accuracy (errors of 10^10 and below) and produced these results up to 10 times faster using a very small population of 50 genotypes (Whitley 1989).

Creating Selective Pressure

First, it should be noted that while more complex mechanisms for selective pressure could possibly be defined, a single variable is adequate. Over the past three year the *GENITOR* project has experimented with several modes of selective pressure, but we have found that the following mechanism works as well as any we have tried and has the advantage of being simple. .ti 3n For a bias up to and including 2.0, a linear function is used to allocate reproductive trials. (See Appendix One.) In the neural net experiments reported here a selective bias of 1.5 was found to give the best overall results. Selective pressures above 1.5 lead to premature convergence, while values below this failed to drive the search hard enough to achieve similar results. A bias of 1.5 implies that the top ranked individual in the population is 1.5 times more likely to reproduce (on one reproductive cycle) than the median individual in the population. For selective pressures greater that 2.0 a nonlinear allocation of trials is used. For example, a selective pressure of 5.0 implied that 5% of all reproductive opportunities go to the top ranked position in the populations. 5% of the remaining 95% of the reproductive opportunities are given the second ranked position, etc. Any residual opportunities are evenly distributed.

Conclusions

Several major conclusions concerning selective pressure are suggested.

(1) The value returned by an evaluation function should not be considered an "exact" measure of fitness. The exact value returned can vary greatly depending on how the function was implemented. There is simply no reason to believe that the value returned by an evaluation function produces appropriate selective pressure, and problems with "super individuals," stagnating searches, and premature convergence provide ample evidence that it does not.

(2) Allocating reproductive trials according to rank prevents scaling problems, since ranking automatically introduces a uniform scaling across the population.

(3) Allocating reproductive trials according to rank provides a means for directly controlling selective pressure. Other parameters such as those used with the *Genesis* package affect selective pressure indirectly and imprecisely.

(4) Ranking, coupled with one-at-a-time reproduction gives the search greater focus. Once the G*ENITOR* algorithm finds a good genotype it stays in the population until displaced by a better string. This means that the algorithm is less prone to "wander" in the search space and will maintain an emphasis on the best schemata found so far. This also removes the need for "Probablistic Crossover."

(5) Allocating reproductive trials according to rank can be used to speed up genetic search. We are currently further testing the use of specialized mutation and increased selective pressure to produce faster yet more accurate optimization. Selective mutation can help sustain genetic diversity, which in turn allows a higher level of exploitation to be achieved via higher selective pressure (Whitley 1989).

ACKNOWLEDGMENTS

This research was supported in part by a grant from the Colorado Institute of Artificial Intelligence (CIAI). CIAI is sponsored in part by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development.

APPENDIX ONE: A Linear Function for Selective Pressure

For selective pressure between 1.0 (random) and 2.0 the following linear algorithm is used. The function "random()" returns a random fraction between 0 and 1.

```
linear()
{ float bias = 1.5;  int index;  double sqrt;

  index = POPULATION_SIZE *
    (bias -
      sqrt (bias * bias - 4.0 (bias -1) * random()))
    / 2.0 / (bias -1);

return (index); }
```

REFERENCES

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

[Baker 1985] Baker J. Adaptive selection methods for genetic algorithms, in: John Grefenstette (Ed.), Proc. of an International Conf. on Genetic Algorithms and Their Applications. (L. Erlbaum, 1988, original proceedings 1985) 101-111.

[Booker 1987] Booker, L. Improving search in genetic algorithms, in: Lawrence Davis (Ed.), Genetic Algorithms and Simulated Annealing. (Morgan Kauffmann, 1987) 61-73.

[DeJong 1975] DeJong, K. Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. Dissertation, University of Michigan.

[Grefenstette 1984] J. Grefenstette A user's guide to GENESIS, Tech. Report CS-84-11, Computer Science Dept., Vanderbilt Univ., Nashville, TN.

[Grefenstette 1986] Grefenstette, J., Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics,* SMC-16 (1) (1986): 122-128.

[Holland 1975] Holland, J. Adaptation in Natural and Artificial Systems. (Univ. of Michigan Press, Ann Arbor, 1975).

[Rumelhart 1986] Rumelhart, D., and McClelland, J., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol I (MIT Press, Cambridge, MA 1986).

[Whitley 1988a] Whitley D., and Kauth J. G*ENITOR*: a different genetic algorithm, in Proc. of the Rocky Mountain Conf. on Artificial Intelligence, Denver, CO (1988) 118-130.

[Whitley 1988b] Whitley D., and Kauth J. Sampling Long Schemata in Genetic Algorithms. Tech. Report CS-88-105, Computer Science Dept., Colorado State Univ.

[Whitley 1988c] Whitley D. Applying Genetic Algorithms to Neural Network Learning. Proc. 7th Conf. for the Study of Artificial Intelligence and Simulated Behavior, Sussex, England. Pitman Publishing.

[Whitley 1989] Whitley, D., and Hanson, T. Optimizing Neural Networks Using Faster, More Accurate Genetic Search. 1989 Genetic Algorithm Conference. Morgan Kaufmann, Publishers. In *Proc. International Joint Conference on Artificial Intelligence.*