GENETIC ALGORITHMS AND THEIR APPLICATIONS: Proceedings of the Second International Conference on Genetic Algorithms

John J. Grefenstette
Naval Research Laboratory
Editor

GENETIC ALGORITHMS AND THEIR APPLICATIONS: Proceedings of the Second International Conference on Genetic Algorithms

July 28-31, 1987
at the
Massachusetts Institute of Technology
Cambridge, MA

Sponsored By

American Association for Artificial Intelligence

Naval Research Laboratory

Bolt Beranek and Newman, Inc.

John J. Grefenstette
Naval Research Laboratory
Editor

LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS
1987 Hillsdale, New Jersey Hove and London

Lawrence Erlbaum Associates, Inc., Publishers
365 Broadway
Hillsdale, New Jersey 07642

ISBN 0-8058-0158-8 cloth edition

ISBN 0-8058-0159-6 paperback edition

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ACKNOWLEDGEMENTS

On behalf of the Conference Committee, it is my pleasure to acknowledge the support of our sponsors: the American Association for Artifical Intelligence, the Navy Center for Applied Research in Artificial Intelligence at the Naval Research Laboratory, and Bolt Beranek and Newman, Inc. The Committee also appreciates the cooperation of Dr. Edwin H. Land. I would personally like to thank the other members of the Conference Committee for their conscientious efforts as referees. Stewart Wilson deserves special thanks for handling the local arrangements.

John J. Grefenstette
Program Chair

Conference Committee

John H. Holland	University of Michigan (*Conference Chair*)
Lashon B. Booker	Navy Center for Applied Research in AI
Dave Davis	Bolt Beranek and Newman
Kenneth A. De Jong	George Mason University
David E. Goldberg	University of Alabama
John J. Grefenstette	Navy Center for Applied Research in AI (*Program Chair*)
Stephen F. Smith	Carnegie-Mellon Robotics Institute
Stewart W. Wilson	Rowland Institute for Science (*Local Arrangements*)

CONFERENCE PROGRAM

TUESDAY, JULY 28, 1987

5:00 - 9:00 REGISTRATION: *Lobby, McCormick Hall*

7:00 - 9:00 WELCOMING RECEPTION: *Courtyard, McCormick Hall*

WEDNESDAY, JULY 29, 1987

8:00 REGISTRATION: *Room 10-280*

9:00 OPENING REMARKS: *Room 10-250*

9:20 - 10:40 GENETIC SEARCH THEORY

10:40 - 11:00 COFFEE BREAK

11:00 - 12:00 ADAPTIVE SEARCH OPERATORS I

12:00 - 2:00 LUNCH

10:40 - 12:00 CONNECTIONISM AND PARALLELISM I

12:00 - 2:00 LUNCH

2:00 - 3:20 PARALLELISM II

3:20 - 3:40 COFFEE BREAK

3:40 - 5:00 CREDIT ASSIGNMENT AND LEARNING

THURSDAY, JULY 30, 1987

6:30 - 10:00 CONFERENCE BANQUET: *New England Clambake*

FRIDAY, JULY 31, 1987

9:00 - 10:20 APPLICATIONS I

10:20 - 10:40 COFFEE BREAK

10:40 - 12:00 APPLICATIONS II

12:00 - 2:00 LUNCH

2:00 - 3:20 PANEL DISCUSSION: GA's and AI

3:20 - 3:40 COFFEE BREAK

3:40 - 5:00 INFORMAL DISCUSSION AND FAREWELL

FINITE MARKOV CHAIN ANALYSIS OF GENETIC ALGORITHMS

David E. Goldberg and Philip Segrest

The University of Alabama
Tuscaloosa, AL 35487

ABSTRACT

A finite Markov chain analysis of a single-locus, binary allele, finite population genetic algorithm (GA) is presented in this paper. The Markov analysis is briefly derived and computations are presented for two kinds of problems: genetic drift (no preference for either allele) and preferential selection (one allele is selected over the other). Approximate analyses are presented to explain the detailed Markov analysis. These computations are useful in choosing parameters for artificial genetic search.

INTRODUCTION

Genetic algorithms (GAs) are receiving increasing application in a variety of search and machine learning problems. These efforts have been greatly aided by the existence of theory that explains what GAs are processing and how they are processing it. The theory largely rests on Holland's exposition of schemata (1968, 1975), his bridge to the two-armed bandit problem (1973, 1975), his fundamental theorem of genetic algorithms (1973, 1975), and later works by several of his students (Martin, 1973; De Jong, 1975; Bethke, 1981). All of these works have necessarily made bounding assumptions: population sizes have been assumed to be infinitely large, probability limits have been estimated by relatively crude limits, and in some cases genetic operators have even been modified to facilitate the analysis. As a result, there is still a need for more exact analysis of genetic algorithm behavior using finite populations and realistic analytical models of genetic operators.

In this paper, we perform a Markov chain analysis of a one-locus, two-operator (reproduction and mutation) genetic algorithm acting upon a finite population of haploid binary structures. Specifically, we calculate the expected time of first passage to various levels of convergence under different selection ratios and mutation rates.

To understand the Markov chain analysis and its application, we first develop an analysis of genetic drift: convergence in finite populations under no selective pressure. Understanding this phenomenon is useful in explaining why finite GAs make convergence errors at relatively unimportant bit positions. We then examine expected GA performance at different levels of selective pressure assuming a deterministic fitness function. This analysis permits calculation of the selective pressure required to reduce the probability of selecting the wrong bit to some known level. We also discuss how these results and their extension may be used for designing and sizing finite GAs.

STOCHASTIC ERRORS IN FINITE GENETIC ALGORITHMS

As simple three-operator genetic algorithms have been used in a wider array of search and machine learning applications (Goldberg & Thomas, 1986), a number of objections have been voiced concerning their performance. Chief among these objections is the occurrence of premature convergence (De Jong, 1975). Premature convergence is that event where a population of structures attains a high level of uniformity at all loci without containing sufficiently near-optimal structures. Two reasons have been given for this undesirable form of convergence: a problem (with its chosen coding) can itself be GA-hard, or the finite GA can suffer from stochastic errors.

GAs may diverge because a problem is inherently difficult for a three-operator GA. Such problems have been called GA-hard (Goldberg, 1983), and Bethke (1981) has proved that GA-hard problems exist using Walsh function computation of schema averages. Reordering operators based on natural precedent have been suggested (Holland, 1975; Goldberg & Lingle, 1985) as one remedy for such problems, especially those where building blocks are not linked tightly enough to permit the three-operator GA to find near-optimal points. Despite the possibility of GA-hardness, Bethke's study and an extended schema analysis of the minimal deceptive problem (Goldberg, 1986) suggest that it is more difficult to construct intentionally misleading (GA-hard) problems than was previously thought. Furthermore this notion --that GA-hard problems are relatively hard to construct--is empirically supported by the widespread success of simple GAs across a spectrum of problems. Nonetheless, the study of GA-hard problems and the design of operators to circumvent

the difficulty remain important open areas of research activity.

The other major source of convergence difficulty in genetic algorithms results from the stochastic errors of small populations. These errors may themselves be divided into two types: errors in sampling and errors in selection. A pollster makes a sampling error when he selects a sample size which is too small to achieve the accuracy he desires. He also commits a sampling error when the sample he selects is not representative of the population as a whole. Genetic algorithms may make similar sampling errors when either the strings representing important schemata are not present in sufficient numbers or the individuals present are not representative of the whole similarity subset. Sampling errors in small populations in this way prevent the proper propagation of the correct (above average) schemata, thereby circumventing the expected and desired action predicted by the schema theorem (Holland, 1975).

Errors of selection are harder to understand because their disruptive effects are counterintuitive. A simple example will help clarify the problem. Suppose we have a population of 50 single-locus structures containing 25 ones and 25 zeros. Suppose further that we select a new population from the old population by choosing 50 new members one at a time using selection with replacement (where once picked, an individual is placed back in the sampled population). Since we are picking each new member with probability 1/50, and since there are currently 25 ones and 25 zeros, the expected number of ones and zeros is still 25 and 25 respectively; however, because the selection process is fairly noisy, we shouldn't expect to retain exactly the same 25/25 split. In fact, the probability of exactly that division is only $P(25/25) = \binom{50}{25}(0.5)^{50} = 0.1123$. Therefore it is reasonably likely the population will fall away from this initial starting point. In the next generation, the new population becomes the new initial condition with the expected number of ones and zeros determined by the new state. This process continues, and in finite time for a finite population, the population converges to all ones or all zeros. Once converged there is no way to get back any of the missing material (unless we permit some mutation, a possibility we will consider later).

Geneticists have long recognized that finite populations do converge in this manner even when there is no selective advantage for one allele over another. This error of selection is so important it has been given a special name, genetic drift. Selection errors can accumulate, causing a drift to one allele or the other. By itself, genetic drift is bad enough. After all, if the environment has no preference for one allele or another, we might like a GA to preserve both of them (perhaps we would even like the GA to preserve them in relatively equal numbers). Genetic drift insures that this won't happen unless we take special action to encourage this behavior (see Goldberg & Richardson, this volume). To make matters worse, these errors of selection can cause a genetic algorithm to converge to the wrong allele when the environment does prefer one allele over the other, especially when that selective advantage is relatively small. The analysis of these difficulties is our main concern in the remainder of this paper. We consider first the case of no selection pressure--genetic drift --followed by analysis of cases with varying degrees of selective pressure.

MARKOV CHAIN ANALYSIS OF GENETIC DRIFT

Discrete and continuous models of genetic drift have received attention in the literature of mathematical biology (Crow & Kimura, 1970); however, many of these analyses contain additional details of biological reality that are not always of interest to genetic algorithmists. De Jong (1975) presents computer simulations of genetic drift in the context of simple GAs, showing graphs of the relationship between expected first passage time to varying convergence levels as a function of population size and mutation rate. In this section, we calculate these quantities more exactly using the mathematics of Markov chains (Kemeny & Snell, 1960).

Markov Chains

Suppose we have a sequence of random variables $s_0, s_1, \ldots$, and suppose the possible values for these random variables are drawn from the set $\{0, 1, \ldots, N\}$. We think of the random variables s_t as the state of some system at time t: more precisely, the system is in state i at time t if $s_t = i$. If at each time t there is a fixed probability p_{ij} that the system will be in state j at time t+1 when the system was in state i at time t, we say the sequence of random variables forms a Markov chain. The fixed quantities p_{ij} are said to be transition probabilities:

$$p_{ij} = P(s_{t+1}=j \mid s_t=i)$$

Markov chains may be classified by the types of states they contain. States that may not be reached as a process goes to infinity are said to be transient states. Those that may be reached as time marches on are said to be ergodic states (non-transient). In this paper, we will only be interested in those Markov chains that contain a particular type of ergodic state: an absorbing state. A state is said to be absorbing if once entered it may never be left. For any state i, this is true if and only if the following condition holds:

$$p_{ii} = 1$$

We have already seen examples of absorbing states in our earlier description of genetic drift: we recognize the all-zeros and all-ones states as absorbing states, because once we get to either one of them we can never leave. A chain with at least one absorbing state and no ergodic states

other than absorbing states is said to be an absorbing Markov chain.

The states of an absorbing Markov chain may be renumbered to obtain a transition probability matrix P in the following canonical form (Kemeny & Snell, 1960):

$$P = \left(\begin{array}{c|c} I & 0 \\ \hline R & Q \end{array}\right)$$

where I is an identity matrix and 0 is a matrix of all zeros. The submatrices Q and R are used to calculate important properties of an absorbing chain.

Genetic Drift without Mutation

To illustrate the construction of a transition probability matrix, we turn to the genetic drift problem at hand. Suppose we have a population of N ones and zeros. We let state 0 be that situation where we have all zeros and we let state N be that case where we have all ones (no zeros), and in general we let state i be that state with exactly i ones. Thus we have a total of N+1 states, $i = 0, 1, \ldots, N$, that together represent all possible conditions within the population. Then if we assume random selection of exactly N new population members with replacement, we may calculate the elements of our drift transition probability matrix P_d as follows:

$$P_d = (p_{ij})_{drift} = \binom{N}{j}\left[\frac{i}{N}\right]^j\left[1-\frac{j}{N}\right]^{N-j}$$

These calculations are provably correct, because at each state i, we have a probability of selecting a one $p_{one} = i/N$; further, the probability of getting to state j (j ones) is binomially distributed with probability p_{one} and size N. In this matrix, the states 0 and N are both absorbing states because $p_{00} = p_{NN} = 1$. Thus, it is a simple matter to obtain the canonical form of the matrix shown above. In particular, we are interested in the Q matrix which may be obtained simply by stripping off the first and last rows and columns of the original P_d matrix.

The Q matrix may then be used to calculate the expected number of visits n_{ij} to any transient state j starting in the transient state i (the N matrix):

$$N = (I - Q)^{-1}$$

We won't belabor the details leading to this calculation here; however, the desired N matrix may first be written as an infinite matrix geometric series in Q. Thereafter, the closed form calculation may be obtained in a manner similar to that used in summing an infinite scalar geometric series.

The total expected time in transit to any one of the absorbing states starting in state i--we call this the transit time τ_i--may easily be calculated as a row sum over the ith row of the N matrix:

$$\tau_i = \sum_{j=0}^{N} n_{ij}$$

The τ_i values may then be used to calculate the expected time to absorption given any initial starting conditions. If we assume an initial population chosen uniformly at random, the probability of being in a state i initially--we call this π_i--is binomially distributed as follows:

$$\pi_i = \binom{N}{i}\left(\frac{1}{2}\right)^N$$

The expected time to an absorbing state, the quantity $t_{absorbed}$, may then be calculated as the dot product of the initial state probability vector π_i and the transit time vector τ_i:

$$t_{absorbed} = \sum_{i=0}^{N} \tau_i \pi_i$$

These calculations suggest a straightforward procedure for calculating the expected time until either the all-ones state (state N) or all-zeros state (state 0) is reached, but what can we do if we are interested in calculating the time of first passage to some level of convergence other than 100 percent? We may calculate such quantities quite simply by replacing all rows in the P_d matrix within a desired percentage of convergence by identity rows: $p_{ii} = 1$ and $p_{ij} = 0$ for $i \neq j$. This bit of chicanery makes the once-transient states absorbing states, and the procedure described above may be used on a now-reduced Q matrix to determine expected first passage times to the particular level of convergence.

These computations are carried out for convergence levels of 60%, 70%, 80%, 90%, and 100%. The expected time of first passage is calculated for populations ranging from N=10 to N=100. Computations have been performed in APL to exploit that language's facility with vector and matrix operations. Figure 1 displays the expected first passage time versus population size at different convergence levels. The relationship between passage time and population size at a given level of convergence is strikingly linear. Although we may expect the passage time to grow with increased population size, it is not at all obvious why that growth is linear.

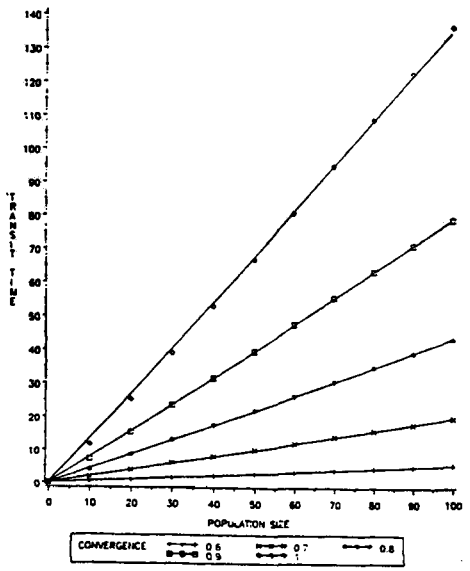


Figure 1. Genetic drift Markov chain computation of expected first passage time versus population size at different convergence levels with no mutation.

To understand this better, we appeal to a simpler but related stochastic model: the gambler's ruin problem. In the gambler's ruin problem, a gambler with a fixed stake D places successive one dollar wagers on the outcome of a coin toss. In our version of the game we assume that the coin is fair, $p_{winning} = p_{losing} = 0.5$. The game ends when the gambler either loses all his money or attains a goal value value A. All this is interesting, but how is it at all related to the genetic drift problem? We may view the genetic drift problem as a gambler's ruin problem where we drift toward either ruin (all zeros) or reward (all ones). In the real genetic drift problem, our range of outcomes on a given "gamble" is not binary; however, by using an average step size per generation we may calculate an approximate form for the genetic drift solution using the simpler gambler's ruin model.

Fortunately, the expected duration of the gambler's ruin problem is a well-known computation (Feller, 1968). Under the assumptions above, the expected number of coin tosses to ruin or reward may be calculated as E(number of coin tosses) = D(A-D). In our problem, the stake amount is given by $N(p_{con}-0.5)$ (the distance to ruin or reward at a particular convergence level, p_{con}); the desired accumulation to quit (assuming equal distances) is twice the stake amount. To place these quantities in terms of numbers of generations, we divide both terms (D and A-D) by the average number of steps taken per generation. This quantity is simply the standard deviation of the expected number of ones (or zeros) per generation:

$$\sigma = \sqrt{Np_{con}(1-p_{con})}$$

This operation yields the following approximate expression for the number of generations to a particular level of convergence:

$$t_{convergence} \propto \left[\frac{(2p_{con}-1)^2}{4p_{con}(1-p_{con})} \right] N$$

Thus we have reasoned that the expected transit time increases linearly with population size. As with many approximate models, the derived proportionality constant is mainly useful an indicator of order of magnitude. The simple model does help explain why the slopes of lines of successively higher convergence proportion increase faster than the proportion itself.

Genetic Drift with Mutation

To combat the undesired convergence of genetic drift in simple GAs, mutation rate increases are often suggested. To investigate the effect of mutation on expected times of convergence we continue our Markov chain analysis by including mutation in our overall probability transition matrix.

With mutation included, we view the overall probability transition matrix P_o as the product of the drift matrix P_d developed earlier and a mutation probability transition matrix P_m:

$$(P_o) = (P_d)(P_m)$$

To develop the mutation transition probability matrix, we need to select from a number of possible mutation models. We may perform a single mutation with probability p_m, replace the individual in the original population, and then perform a sequence of N such operations (we call this mutation with sequential replacement). We may also mutate (again with probability p_m) population member by population member (mutation without replacement). We develop the transition probability matrices for both types of mutation.

The transition probability matrix for mutation with sequential replacement may be developed quite simply. A single mutation permits a shift in state by one step (either one more one or one less one). Thus, the single mutation transition probability matrix P_{ms} is tridiagonal, and its elements may be specified as follows:

$$p_{i(i+1)} = \left(1 - \frac{i}{N}\right)p_m$$

$$p_{ii} = 1-p_m$$

$$P_{i(i-1)} = \left(\frac{i}{N}\right)p_m$$

$$P_{ij} = 0, \quad \text{otherwise}$$

The overall mutation transition probability matrix under sequential replacement may then be taken as the product of N single mutation matrices:

$$(P_m)_{\text{sequential replacement}} = (P_{ms})^N$$

The transition probability matrix for mutation without replacement may also be calculated. Although the computation is more cumbersome, the extra effort is worthwhile as mutation without replacement is a more faithful model of mutation as it is implemented in most GAs. To calculate the transition probability matrix for mutation without replacement, we recognize a simple fact. To go up in state value by, for example, two ones, we must have exactly two more zeros change to ones than we have ones change to zeros. If we sum over all the possible occurrences where this is true we obtain the elements of the transition probability matrix. For $j \geq i$ (where we shift to a state with more ones) we obtain the elements of the transition probability matrix as follows:

$$P_m = (P_{ij})_{\text{mutation no replacement}}$$

$$= \sum_{k=0}^{\ell} \binom{i}{k}\binom{N-i}{k+j-i} p_m^{2k+j-i}(1-p_m)^{N-2k-j+i}$$

where $\ell = \min(i, N-j)$

A similar expression may be derived for the case where $j \leq i$ (a transition occurs to a state with fewer ones). Because mutation without selection is a better model of mutation as performed in real GAs, we adopt it in all subsequent computations.

We perform similar calculations as in the case without mutation. Expected passage times are plotted versus population size (log-log scale) at different mutation rates in Figures 2 and 3. These graphs are presented for convergence levels of 100% and 80% respectively. Increasing mutation probabilities increase the expected time of first passage to a particular level of convergence as we expect.

CONVERGENCE UNDER PREFERENTIAL SELECTION

The genetic drift cases demonstrate the problem of undesirable convergence when the environment has no preference for one allele over another. In this section, we examine the rate of convergence when there is a preference for a particular allele. We also calculate the probability that the allele we converge to is the correct allele.

We assume that a one allele receives an environmental reward of f_1 and a zero allele

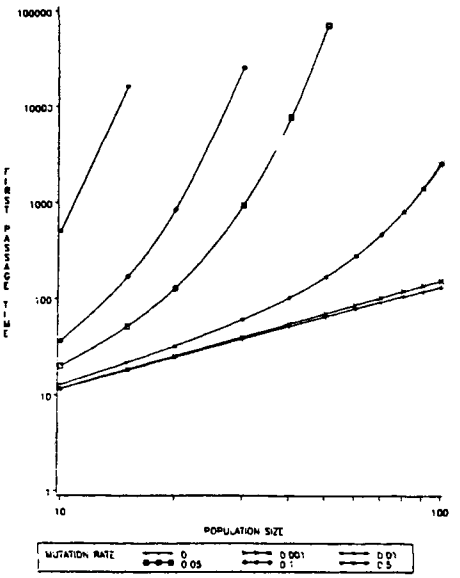


Figure 2. Genetic drift Markov chain computation of expected first passage time versus population size at 100 percent convergence level with different mutation probabilities.

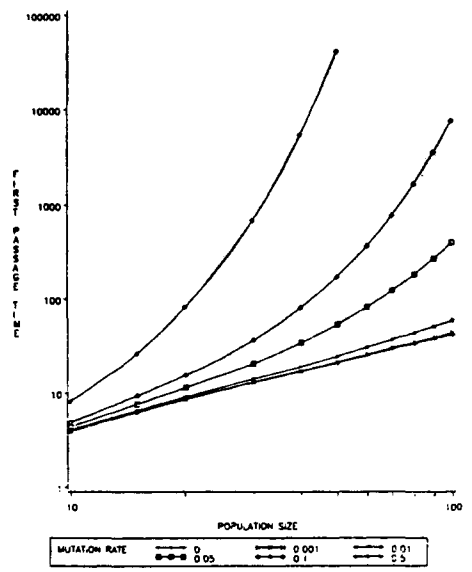


Figure 3. Genetic drift Markov chain computation of expected first passage time versus population size at 80 percent convergence level with different mutation probabilities.

receives an environmental reward of f_0. These rewards are assumed to be stationary and deterministic, and the rewards are given in the ratio $r = f_1/f_0$. In the genetic drift cases of the last section the overall transition probability matrix was the product of the drift matrix and the mutation matrix. Under preferential selection, we must replace the drift

matrix by a reproduction matrix that accounts for the differential pressure toward one allele or another.

We may account for the added pressure toward a particular allele by recognizing that the probability of selecting a particular one is $f_1 / \sum f$, and therefore the probability of selecting a one from the population is the number of ones times the probability of selecting a particular one:

$$p_1 = \frac{i \cdot f_1}{\sum_{\text{all } s} f(s)} = \frac{f_1}{i \cdot f_1 + (N-i) f_0} = \frac{r \cdot i}{r \cdot i + (N-i)}$$

where the sum in the denominator is take over all structures s. With this selection probability, the transition probability matrix for reproduction (P_r) may be calculated:

$$P_r = (p_{ij})_{\text{reproduction}} = \binom{N}{j} \left(\frac{i \cdot r}{i \cdot r + (N-i)} \right)^j \left(\frac{N-i}{i \cdot r + (N-i)} \right)^{N-j}$$

The probability of going from a state with i ones to a state with j ones is binomially distributed with the selection probability a function of the state i and the fitness ratio r. Notice that the reproduction matrix reduces--as it must--to the drift matrix as r goes to one.

Using the same procedure as before, we may calculate the expected time to some level of convergence; however, in the case where the environment is not indifferent to the two alleles, we are not simply interested in knowing when we expect to converge. We are also interested in knowing the probability that we converge to the allele with higher fitness. Fortunately we may obtain this probability in a straightforward manner using our fundamental N matrix and the R matrix.

The probability of starting in a transient state i and going to absorbing state j--we call this the B matrix, (b_{ij})--may be obtained as the product of the fundamental matrix N and the R matrix:

$$B = NR$$

That this is so may be reasoned directly (Kemeny & Snell, 1960). Any time we are in transient state k, we have probability r_{kj} of going to absorbing state j; however we will be in state k assuming we have started in state i a total of n_{ik} times. To finish the computation, we simply realize that we must sum over all the transient states k leading to state j. This is precisely the computation represented by the matrix product above.

With the probability b_{ij} of going from a starting transient state i to a final absorbing state j thus calculated, it is a simple matter to obtain the probability of being in a final state j--the probability α_j--as time goes on assuming initial state probabilities π_i. We simply add the initial probability of being in state j to the sum of the probabilities of getting from all transient states i--call this the set T--to the particular absorbing state j:

$$\alpha_j = \pi_j + \sum_{i \in T} \pi_i b_{ij}$$

Having calculated the probability of ending up in a particular absorbing state, we now may find the probability of convergence to the correct allele, p_{correct}, by summing the α values over the desired absorbing states--call this set A':

$$p_{\text{correct}} = \sum_{i \in A'} \alpha_i$$

We calculate the expected time to 100 percent convergence and the probability that the convergence is correct for varying r values at different population sizes. These results are plotted in Figures 4 and 5 respectively. To put these results in perspective, we consider the expected performance in a large population and compare those results to the exact study of finite populations.

The schema theorem (Holland, 1975) tells us what to expect in a large population. Under reproduction only, the proportion of ones P_1 grows according to the following equation:

$$P_1^{t+1} = \left[\frac{r}{(r-1)P_1^t + 1} \right] P_1^t$$

where the superscript t is a generation index. For r values near 1 this equation reduces to a geometric progression in r:

$$P_1^t = (r^t) P_1^0$$

Taking the natural logarithm of both sides we obtain the following relationship for the time to a given proportion of ones:

$$t = \frac{\ln \left(P_1^t / P_1^0 \right)}{\ln r}$$

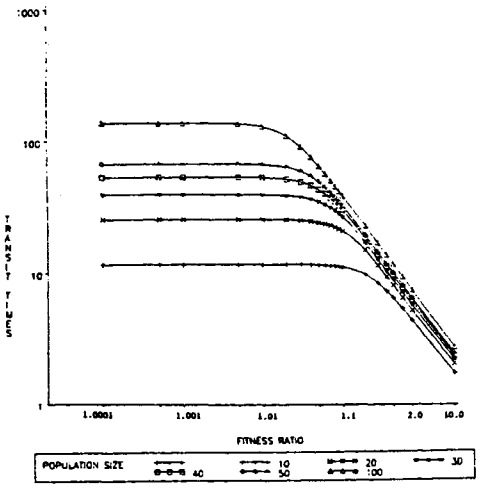


Figure 4. Markov chain computation of expected first passage time versus fitness ratio r for reproduction cases to 100 percent convergence level with different population sizes.

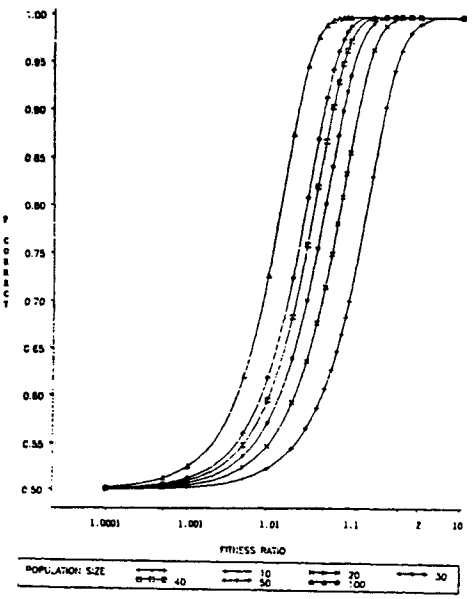


Figure 5. Markov chain computation of probability of correct convergence versus fitness ratio r for reproduction cases to 100 percent convergence level with different population sizes.

On a plot of log(t) versus log(log r) this curve plots as a straight line with negative slope. Referring to Figure 4, at high enough r values we notice this straight line behavior. At small r values the curve levels out and approaches the constant value predicted by the genetic drift computations. We may derive an approximate r value where this divergence should occur with a little physical reasoning. During a given generation we expect an $_t$ increase in ones approximately equal to $N(r-1)P_1^t$. On average, the increase or decrease of ones due to selection noise is simply the standard deviation σ. When the noise is of the same order of magnitude or greater than the expected increase, we should expect divergence between simplified model and the finite model. This divergence should occur at values of r predicted by the following relationship:

$$r-1 \leq \sqrt{\frac{1-P_1^t}{NP_1^t}}$$

When the population is half ones and half zeros, this equation says that the large population result becomes suspect when the excess fitness (r-1) is less than the inverse of the square root of the population size. This relationship explains the decreasing break point value (between drift-like and convergent behavior) with increasing population size as can be seen in both Figures 4 and 5. At values of r beneath this critical value, the expected time results flatten, and the probability of converging to the correct allele starts to fall dramatically.

Similar calculations may be performed for reproduction with mutation. As in the genetic drift case, we take the overall transition probability matrix as the product of two matrices; here, we multiply the reproduction transition probability matrix by a mutation matrix (without replacement) as follows:

$$(P_o) = (P_r)(P_m)$$

We perform expected first passage time calculations as before. In Figure 6, we graph first passage time versus r for different mutation probabilities p_m at a population size N=50. Using our previous physical analysis, we expect the finite analysis to approach the bounding analysis for r-1 values greater than sqrt(1/50) = 0.1414. This holds true for small p_m values; however, as the mutation probability grows, the expected time grows beyond that predicted by the bounding analysis. That this should occur may be reasoned intuitively. If the expected net loss of good alleles due to mutation starts to exceed the expected increase in good alleles from reproduction, the process loses its ability to converge as quickly. This point occurs when the following condition holds true:

$$r-1 \leq \left[\frac{2P_1^t-1}{P_1^t}\right]p_m$$

When the excess fitness (r-1) is of the same order or less than the mutation, the GA will take a longer time to converge, because mutation is adding errors faster than the reproduction can erase them. This shows up in Figure 6 as succes-

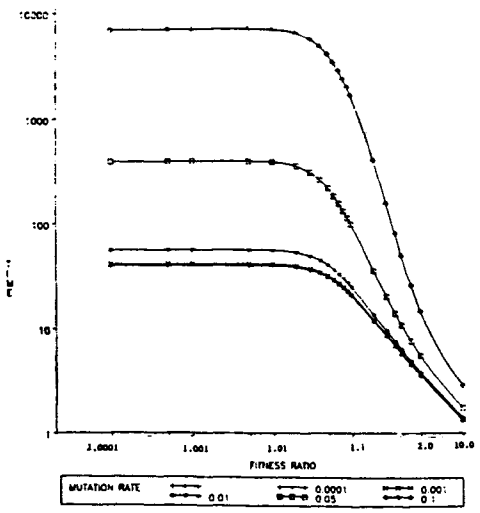


Figure 6. Markov chain computation of expected first passage time versus fitness ratio r for reproduction and mutation cases to 90 percent convergence with N=50 and different mutation levels.

sively higher mutation rates cause the expected time curves to break away from the bounding analysis results more quickly.

CONCLUSIONS

In this paper, we have analyzed the performance of one-locus, two-operator (reproduction and mutation) genetic algorithms with finite populations of haploid structures using finite Markov chains.

The results in particular and the Markov chain analysis in general are useful in understanding the performance of finite GAs commonly in use. These results and their extension should be useful in sizing populations appropriately, selecting proper mutation rates, and choosing rates of selection in scaling procedures.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant MSM-8451610.

REFERENCES

Bethke, A. D. (1981). Genetic algorithms as function optimizers. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 41(9), 3503B. (University Microfilms No. 8106101)

Crow, J. F., & Kimura, M. (1970). An introduction to population genetics theory. New York: Harper and Row.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 36(10), 5140B. (University Microfilms No. 76-9381)

Feller, W. (1968). An introduction to probability theory and its applications (Vol. 1, 3rd ed.). New York: Wiley.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 44(10), 3174B. (University Microfilms No. 8402282)

Goldberg, D. E. (1986). Simple genetic algorithms and the minimal deceptive problem (TCGA Report No. 86003). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., & Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. In J. J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and Their Applications (pp. 154-159). Pittsburgh: Carnegie-Mellon University.

Goldberg, D. E., & Thomas, A. L, (1986). Genetic algorithms: A bibliography 1962-1986 (TCGA Report No. 86001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Holland, J. H. (1968). Hierarchical descriptions of universal spaces and adaptive systems (Technical Report ORA Projects 01252 and 08226). Ann Arbor: University of Michigan, Department of Computer and Communication Sciences.

Holland, J. H. (1973). Genetic algorithms and the optimal allocations of trials. SIAM Journal of Computing, 2(2), 88-105.

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

Kemeny, J. G., & Snell, J. L. (1960). Finite Markov Chains. Princeton: Van Nostrand.

Martin, N.(1973). Convergence properties of a class of probabilistic adaptive schemes called sequential reproductive plans. (Doctoral dissertation, University of Michigan), Dissertation Abstracts International, B-3747B. (University Microfilms No. 74-3685)

AN ANALYSIS OF REPRODUCTION AND CROSSOVER IN A BINARY-CODED GENETIC ALGORITHM

Clayton L. Bridges and David E. Goldberg

The University of Alabama
Tuscaloosa, AL 35487

ABSTRACT

The foundation of genetic algorithm (GA) theory--the so-called schema theorem or fundamental theorem of genetic algorithms--provides a lower bound on the expected number of representatives of a particular schema (similarity subset) in the next generation under various genetic operators. In this paper, assuming a large population of binary, haploid structures of known distribution, processed by fitness proportionate reproduction, random mating, and random, single-point crossover, an exact expression for the expected proportion of a particular string (or representatives of a particular schema) in the next generation is calculated. This derivation is useful in analyzing the expected performance of simple GAs.

INTRODUCTION

Over the past two decades, the application of genetic algorithms (GAs) to search and machine learning problems in science, commerce, and engineering has been been made possible by a number of theoretical developments (Holland, 1973, 1975; De Jong, 1975; Bethke, 1981). Without these theories, it is doubtful that many of us would have made much sense of our computer simulations or experiments; this speculation is supported by the experience of genetic algorithm prehistory. We need only recall some of the evolutionary schemes that resorted to mutation-plus-save-the-best strategies (Box, 1957; Bledsoe, 1959; Friedman, 1959; Fogel, Owens, & Walsh, 1966) to remember that shots in a darkness without schemata or the fundamental theorem (Holland, 1975) can be frustrating experiences indeed. Because of the usefulness of theory to progress in genetic algorithm research, there is still a pressing need to improve our understanding of the foundations--the theoretical underpinnings--of genetic algorithms and their derivatives.

In this paper, we extend the fundamental theorem of genetic algorithms to exactitude. Specifically, we derive a complete set of equations describing the combined effects of reproduction and crossover on a large population of binary, haploid structures. These equations may be used to determine the correct expected performance of a genetic algorithm on a given problem with specified coding; they may be used for calculating the correct expected propagation of a set of competing schemata; they may also be used for estimating disruption or source probabilities for particular strings or particular schemata in a specified population of structures.

In the remainder of the paper, we develop our extended analysis in three steps. We reexamine the fundamental theorem of GAs (the schema theorem), calculate an exact expression for the probability of disruption due to crossover, and calculate the expected gain of individuals due to mating and crossover by others.

THE FUNDAMENTAL THEOREM OF GENETIC ALGORITHMS

The fundamental theorem of genetic algorithms (Holland, 1975) calculates a bound on the expected number, m, of schemata (similarity templates), H, in successive generations, t, under the action of reproduction and crossover (other operators are often included in the calculation; we choose to focus on reproduction and crossover alone):

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1 - p_c \cdot \frac{\delta(H)}{\ell-1}\right]$$

In this equation, ℓ is the string length, p_c is the probability of crossover, and δ is the schema defining length (the distance between its outermost defining positions). In words, the schema theorem tells us that a particular schema H receives trials according to the ratio of schema fitness to population average fitness as long as the schema is not unduly disrupted by crossover. The average fitness of a schema f(H) may be calculated by summing over all strings s_i representatives of H at time t:

$$f(H) = \frac{\sum_{(i|s_i(t)\in H)} f_i}{m(H,t)}$$

We note that the theorem is an inequality--a lower bound. Our main goal in this paper is to transform the bound to an equality. To do this, it is helpful

to look at a full schema conservation equation in broad outline:

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}}\left[\ 1-p_c p_d\ \right] + [\text{gains from crossing}]$$

The increase due to reproduction is multiplied by the probability of the schema surviving a cross (one minus the product of the crossover probability p_c and the probability of disruption p_d). For completeness we must also include possible gains from mating and crossover of other strings.

Hidden within this broad outline are the two items that prevent the schema theorem from providing an exact analysis. First, we usually calculate a crude upper bound on the probability of disruption p_d due to crossover: the usual term $\delta(H)/(\ell-1)$ assumes that the schema is destroyed every time a cross falls within its defining length. This is a conservative assumption because it ignores the possibility of getting back the same material from a particular mate. Second, the schema theorem ignores all sources of schemata from crosses of strings containing different competing schemata. In crossover, one schema's loss is another's gain. Although these terms may seem small and somewhat beside the point, a recent study (Goldberg, 1986) has shown that inclusion of these terms in an extended schema analysis permits the prediction of convergence of a simple genetic algorithm on a problem specifically designed to cause the GA to diverge (the minimal, deceptive problem). As such, these terms deserve more of our attention. We first look at the probability of disruption due to crossover.

DISRUPTION DUE TO CROSSOVER

In this section we calculate the probability of disruption of a string due to the set of all possible crosses. To do this, we define some useful symbols and terms.

We take S as the set of all possible binary strings of length ℓ; there are 2^ℓ such strings. Furthermore, we index the set S so S_j is the the jth member of S, where j goes from 0 to $2^\ell-1$. As a convenience, we choose our indexing scheme so the decoded value of our string is equal to the index value j. We note that we are working in terms of strings, and the schema theorem is written in terms of schemata. We continue here to work with strings as the equations are easier to derive and understand; however, keep in mind that we ultimately want to come back to a particular set of competing schemata. We do this in a later section.

We define arbitrary string variables B, Q, and R; these may take on any value in S. We also permit ourselves to refer to their individual positions by subscripted lower case letters:

$$B = b_0 b_1 \ldots b_{\ell-1}$$

Thus b_j is the boolean variable representing the jth position of a string of length ℓ.

Before we proceed with the derivation of the probability of disruption, we convert the schema theorem to proportion form. Suppose reproduction is operating by itself. In this case, we only have the first part of the schema theorem:

$$m(B,t+1) = m(B,t)f(B)/\bar{f}.$$

If we divide both sides of the equation by the population size N, and if we define P_B^t to be the proportion of the string B in the population at time t, we obtain the following equation for the expected proportion in the next generation (or the mating pool) under the action of reproduction alone:

$$P_B^{t+1} = P_B^t \cdot \frac{f_B}{\bar{f}}$$

where f_B is the fitness of string B and $\bar{f}$ is the average fitness of the population as given by the following expression:

$$\bar{f} = \sum_{i=0}^{2^\ell-1} P_i^t f_i$$

Of course, we are currently interested in calculating the crossover disruption term that reduces the above expression by a factor $1-p_c p_d$ where p_c is the probability of crossover and p_d is the probability of disruption; however, we envision the GA acting in two phases: a selection phase followed by a mating and crossover phase. Therefore it is useful to define the expected proportion of string B under reproduction alone. We shall use this quantity in our calculation of the probability of disruption p_d. We call this important intermediate quantity R_B^t the reproductive proportion:

$$R_B^t = P_B^t \frac{f_B}{\bar{f}}$$

To calculate the probability of disruption to a given string B under crossover, we need to know which strings will disrupt B and which won't. Clearly a string that is different from another string by a single bit cannot fail to return a copy of the original string among the two strings produced by the cross. For example, suppose we have two strings B and B' which differ at position 3 (the bar is used to indicate a complementary bit position):

$$B = b_0 b_1 b_2 b_3 b_4$$
$$B' = b_0 b_1 b_2 \bar{b}_3 b_4$$

Every possible cross produces a copy of B. On the

other hand, strings with two or more bits of difference must (for at least one cross site) disrupt one another. Consider the two strings with three different bits:

$$B = b_0 b_1 b_2 b_3 b_4$$
$$B' = b_0 \bar{b}_1 \bar{b}_2 \bar{b}_3 b_4$$

In this case, disruption occurs if a cross falls between the two outermost bits of difference (between positions 1 and 3). These two outermost bits--we call them sentry bits--are important in the analysis of disruption, because we need only consider their location when deciding how two strings will disrupt one another. We can use this information to create a general scheme for analyzing mutual disruption:

Region	begin	middle	end
Length	x	δ+1	ℓ-δ-x-1
Characteristics	-	$\bar{b}$*...*$\bar{b}$	-

Here we have divided the string into three regions: the beginning, the middle, and the end. In the beginning and ending regions, the strings under consideration must have the same bit values as string B. The middle region is bounded by sentry bits (the b's) where the string B is different from its prospective mate. Bit positions other than the sentry positions are marked with *'s, the usual don't care or wild card symbol used in schema analysis; we may properly use the don't care symbol here, because it really does not matter where we cross between sentry bits. Every such cross is disruptive. To quantify the disruption we define a number of useful variables. We take δ ($\delta \epsilon [1, \ell-1]$) as the defining length of the middle region; this is also the number of possible cross sites. Separately we recognize that the length of the middle section (including both sentries) is δ+1. We define x ($x \epsilon [0, \ell-\delta-1]$) as the length of the beginning region; it also is the position of the first sentry bit. Finally, we recognize that the length of the final section is the remaining portion that causes the total to sum to ℓ: ℓ-δ-x-1.

To facilitate our disruption computation, we define a middle function M = M[B,δ,x]. The middle function is a subset generator, generating a similarity subset according to the following schema:

$$M[B,\delta,x] = b_0 \ldots b_{(x-1)} \bar{b}_x * \ldots * \bar{b}_{(x+\delta)} b_{(x+\delta+1)} \ldots b_{(\ell-1)}$$

We may now use the middle function M to calculate the probability of disruption of a given string B. This probability is the sum over all strings of the product of proportions and defining lengths:

$$p_d(B) = \left[\sum_{\delta=1}^{\ell-1} \frac{\delta}{\ell-1} \cdot \sum_{x=0}^{\ell-\delta-1} \sum_{(j \mid S_j \epsilon M[B,\delta,x])} R_j^t \right]$$

The use of the middle function M insures that we count each string only once. This may be confirmed by removing the δ and R factors and simply using the summation to count the number of such crosses. After clearing the expression, we find that there are 2^{ℓ}-ℓ-1 crosses that match the M function. This quantity may be reasoned independently as the total number of strings less the number of one-bit mismatches (these don't disrupt) and the string B itself (a zero-bit mismatch).

STRING GAINS FROM CROSSOVER

Just as we are interested in detailing the losses from crossover, so are we interested in knowing the potential gain from other string crosses. We may construct a template to precisely identify when this occurs. Specifically, we consider the construction of a string B from two strings Q and R as follows:

Region	begin	middle	end
Length	α	σ	ω
Q Characteristics	*...*$\bar{b}$	-	-
R Characteristics	-	-	$\bar{b}$*...*

We now demand that the Q string be identical to B in the middle and end regions, and we require the R string to be identical to B in the beginning and middle sections. This time around we post sentries outside the middle region, and when we cross between the sentries we obtain a copy of the desired string B as one of the products of the cross. To make our summation a bit easier we define several important quantities.

The quantity α ($\alpha \epsilon [1, \ell-1]$) is the length of the beginning region in string Q. The quantity ω ($\omega \epsilon [1, \ell-\alpha]$) is the length of the ending region in R, and σ ($\sigma = \ell-\alpha-\omega$) is the length of the middle region.

We define formal string functions to specify the strings they may cross to yield B. We call the two functions the beginning function A[B,α] and the ending function Ω[B,ω]. The beginning function is a subset generator as follows:

$$A[B,\alpha] = **\ldots**\bar{b}_{(\alpha-1)} b_\alpha \ldots b_{(\ell-1)}$$

The ending function generates subsets as follows:

$$\Omega[B,\omega] = b_0 \ldots b_{(\ell-\omega-1)} \bar{b}_{(\ell-\omega)} **\ldots**$$

The probability that these two strings will cross to give B is dependent on the value of σ, the length of the region where they are the same. If $\sigma = 0$, then there is only one site where the strings can cross to yield B. In general there are a total of σ+1 cross sites; therefore, the probability that a cross will give B is given by the expression:

$$\frac{o+1}{\ell-1}$$

We may now calculate the expected proportion gain P_g of strings B from crosses by all other strings as follows:

$$P_g(B) = \frac{1}{2}\binom{2}{1} p_c \cdot \sum_{\alpha=1}^{\ell-1} \sum_{w=1}^{\ell-\alpha} \frac{o+1}{\ell-1} \left[\sum_{\{j|S_j \in A[B,\alpha]\}} R_j^t \left[\sum_{\{k|S_k \in \Omega[B,w]\}} R_k^t \right] \right]$$

We multiply by 2 because there are two ways to pick Q and R. We divide by two because only half the products of the cross are B strings.

THE COMPLETE EQUATION

With both disruptions and gains now calculated, it is a straightforward matter to write the expected proportion of strings B in generation t+1 under both reproduction and crossover:

$$P_B^{t+1} = R_B^t \left[1 - p_c \left[\sum_{\delta=1}^{\ell-1} \frac{\delta}{\ell-1} \cdot \sum_{x=0}^{\ell-\delta-1} \sum_{\{j|S_j \in M[B,\delta,x]\}} R_j^t \right] \right] + p_c \left[\sum_{\alpha=1}^{\ell-1} \sum_{w=1}^{\ell-\alpha} \frac{o+1}{\ell-1} \left[\sum_{\{j|S_j \in A[B,\alpha]\}} R_j^t \left[\sum_{\{k|S_k \in \Omega[B,w]\}} R_k^t \right] \right] \right]$$

EXTENSION TO SCHEMATA

So far we have confined our analysis to individual strings. We often want to examine the expected propagation of a set of competing schemata over some specified bit positions. To do this in our current scheme requires only minor modifications to our equations.

To interpret the extended equations in schemata, we first index the sums over the o fixed positions (where o is the schema order or the number of fixed positions) instead of the ℓ positions in the string (we do use ℓ in the denominator of the disruption probability, however). Next, we introduce a function $\Delta(x,\delta)$ to account for the possibly unequal spacing of the fixed positions. For example, if we are interested in the order 3 competing schemata defined by the template f***f*f (where the f indicates a fixed position, a 0 or a 1) at x=0 and δ=1, the defining length function may be calculated as Δ=4-0=4. Since the x values only run through the consecutive fixed positions, at x=1 (the middle fixed position) we find $\Delta(1,1)$=2. With such a function defined, we may rewrite our disruption probability for a schema H as follows:

$$p_d(H) = \left[\sum_{\delta=1}^{o-1} \sum_{x=0}^{o-\delta-1} \frac{\Delta(x,\delta)}{\ell-1} \sum_{\{j|S_j \in M[H,\delta,x]\}} R_j^t \right]$$

Here we interpret S_j as a member of the set of schemata sharing the same o fixed positions as H. Also, R_j is interpreted as the reproductive proportion of the schema S_j.

Thus we are able to extend the computation of the probability of disruption to schemata using the same middle function M through careful indexing and the introduction of a function that tracks unequal defining bit spacing. Similar modification may be introduced in the expected gain proportion computation, P_g.

CONCLUSIONS

In this paper, we have derived an equation for the expected propagation of strings in a genetic algorithm under reproduction and crossover, and we have shown how such a derivation can be extended to schemata. The computation is exact within the assumptions of the analysis (large populations, uniformly random mating, and randomly chosen cross sites). The equations may be used for the analysis of specific problems and codings or it may be used to quantify disruption in populations of strings or schemata of known distribution. The equations should further our understanding of the detailed operation of genetic algorithms.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant MSM-8451610.

REFERENCES

Bethke, A. D. (1981). Genetic algorithms as function optimizers. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 41(9), 3503B. (University Microfilms No. 8106101)

Bledsoe, W. W. (1961, November). The use of biological concepts in the analytical study of systems. Paper presented at the ORSA-TIMS National Meeting, San Francisco, CA.

Box, G. E. P. (1957). Evolutionary operation: A method for increasing industrial productivity. Applied Statistics, 6(2), 81-101.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 36(10), 5140B. (University Microfilms No. 76-9381)

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial intelligence through simulated evolution. New York: John Wiley.

Friedman, G. J. (1959). Digital simulation of an evolutionary process. General Systems Yearbook, 4, 171-184.

Goldberg, D. E. (1986). Simple genetic algorithms and the minimal deceptive problem (TCGA Report No. 86003). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. SIAM Journal of Computing, 2(2), 88-105.

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

REDUCING BIAS AND INEFFICIENCY IN THE SELECTION ALGORITHM

James Edward Baker

Vanderbilt University

Abstract

Most implementations of Genetic Algorithms experience sampling bias and are unnecessarily inefficient. This paper reviews various sampling algorithms proposed in the literature and offers two new algorithms of reduced bias and increased efficiency. An empirical analysis of bias is then presented.

1. Introduction

Genetic Algorithms (GAs) cycle through four phases[4]: evaluation, selection, recombination and mutation. The selection phase determines the actual number of offspring each individual will receive based on its relative performance. The selection phase is composed of two parts: 1) determination of the individuals' expected values; and 2) conversion of the expected values to discrete numbers of offspring. An individual's expected value is a real number indicating the average number of offspring that individual should receive. Hence, an individual with an expected value of 1.5 should average, 1 1/2 offspring. The algorithm used to convert the real expected values to integer numbers of offspring is called the sampling algorithm. The sampling algorithm must maintain a constant population size while attempting to provide accurate, consistent and efficient sampling. These three goals lead to the bias, spread and efficiency measures described below:

1) bias -- Bias is defined to be the absolute difference between an individual's actual sampling probability and his expected value[3]. To ensure proper credit assignment to the represented hyperplanes, the expected values should be sampled as accurately as possible. The optimal, zero bias is achieved whenever each individual's sampling probability equals his expected value.

2) spread -- Let f(i) be the actual number of offspring individual i receives in a given generation. We define the "spread" as the range of possible values for f(i). Furthermore, we define the "Minimum Spread" as the smallest possible spread which theoretically permits zero bias. Hence, the Minimum Spread is one in which

$$f(i) \in \{ \lfloor ev(i) \rfloor, \lceil ev(i) \rceil \}$$

where $ev(i)$ = expected value of individual i.

Whereas the bias indicates accuracy, the spread indicates precision. Hence the spread reveals the sampling algorithm's consistency.

3) efficiency -- It is desirable for the sampling algorithm not to increase the GAs' overall time complexity. GAs' other phases are O(LN) or better, where L = length of an individual and N = population size.

All currently available sampling algorithms fail to provide both zero bias and Minimum Spread[3]. Yet an accurate sampling algorithm is crucial. All of the GAs' theoretical support presupposes the ability to implement the intended expected values. How much the existing inaccuracies have affected the GAs' performance is unknown. However, inaccuracies of sufficient magnitude to alter performance should be either understood or eliminated. This paper offers an empirical analysis of one of the most common sampling algorithms[5] -- Remainder Stochastic Sampling without Replacement--and introduces two new sampling algorithms designed to reduce or eliminate these inaccuracies.

Section 2 reviews and compares previously proposed sampling algorithms. Section 3 introduces an improved sampling algorithm which can be partially executed in parallel. Section 4 introduces a sampling algorithm of zero bias, Minimum Spread and optimal O(N) time complexity. Section 5 presents an empirical analysis of these algorithms and section 6 provides a summary and conclusions.

2. Previous Work

The basic characteristics of various sampling algorithms[3] are presented in Table 1. The first four algorithms are stochastic and involve basically the same technique:

— Determine R, the sum of the competing expected values.
— 1-1 map the individuals to contiguous segments of the real number line, [0..R), such that each individual's segment is equal in size to its competing expected value.
— Generate a random number within [0..R).
— Select the individual whose segment spans the random number.
— Repeat the process until the desired number of samples is obtained.

This technique is commonly called the "spinning wheel" method. It is analogous to a gambler's spinning wheel with each wheel slice proportional in size to some individual's expected value. This technique is typically implemented in $O(N^2)$ time, but can be implemented in O(NlogN) by using a B-tree.

In "Stochastic Sampling with Replacement", the "spinning wheel" is composed of the original expected values and remains unchanged between "spins". This provides zero bias yet virtually unlimited spread – any individual with expected value > 0 could be chosen to fill the entire next population. "Stochastic Sampling with partial Replacement" prevents this. In this algorithm, an individual's expected value is decreased by 1.0 each time he is selected. (If the expected value becomes negative, it is set to 0.) This modification provides an upper bound of ⌈expected value⌉ on the spread. However, this upper bound is achieved at the expense of bias. Furthermore, a reasonable lower bound is not provided.

Remainder sampling methods reduce the spread further. A remainder sampling method involves two distinct phases. In the integral phase, samples are awarded deterministically based on the integer portions of the expected values. This phase guarantees a lower bound of ⌊expected value⌋ on the spread. The fractional phase then samples according to the expected values' fractional portions.

In "Remainder Stochastic Sampling with Replacement", the fractional expected values are sampled by the "spinning wheel" method. The individual's fractions remain unaltered between "spins", and hence continue to compete for selection. This sampling algorithm provides zero bias and the greatest lower bound on the spread, yet it provides virtually no upper bound. Any individual with an expected value fraction > 0 can obtain all samples selected during the fractional phase. "Remainder Stochastic Sampling without Replacement", (**RSSwoR**), provides Minimum Spread. This remainder algorithm also uses the "spinning wheel" for the fractional phase. However, after each "spin", the selected individual's expected value is set to zero. Hence, individuals are prevented from having multiple selections during the fractional phase. Unfortunately, although this is a commonly used sampling algorithm, it is biased by favoring smaller fractions[2].

A "Deterministic Sampling" algorithm is suggested and used by Brindle[3]. In this remainder algorithm's fractional phase, the individuals with the largest fractions are selected. The result is a minimum sampling error for each generation, yet a high overall bias. Since overall accuracy in approximating the expected values is crucial to the applicability of the GAs' current theory, high bias is considered unacceptable. Hence, "Deterministic Sampling" is not widely used.

3. Remainder Stochastic Independent Sampling

In a spinning wheel algorithm, selection is based on the relative expected values of those competing. This relativity is unnecessary since the expected values themselves are population normalized. Furthermore, this relativity is causing the bias-spread tradeoff and the O(NlogN) time complexity. (If the selected individuals are not inhibited in future competitions, then there is little spread limitation. However, if they are inhibited, subsequent selection is biased.)

In "Remainder Stochastic Independent Sampling" (**RSIS**), the fractional phase is performed without use of the error-prone "spinning wheel". **RSIS** independently uses each fractional expected value as a probability of selection. This is accomplished by *traversing* the population and stochastically determining whether each individual should be selected:

"C" Code Fragment for **RSIS**

```
/* Integral Phase */
NumSelected = 0;
for (i=0; i<N; i++)
   for ( ; ExpVal[i] >= 1.0; ExpVal[i]--)
   {     SelectInd(i);
         NumSelected++;
   }
/* Fractional Phase */
for (i=0; NumSelected<N; )
{  if (ExpVal[i] > Rand())
   {     SelectInd(i);
         ExpVal[i] = 0;
         NumSelected++;
   }
   if (++i == N) i=0;
}
```

Where Rand() returns a random real number uniformly distributed within the range [0..1).

The code for **RSIS** is noticeably less complex than typical implementations of sampling and much less than O(NlogN) algorithms. Although the fractional phase is potentially an infinite loop, probabilistically it completes after one traversal of the population and is only O(N). Empirical studies indicate that it rarely proceeds beyond the second traversal –

averaging only 0.017% over a variety of expected value distributions. Hence a modified algorithm which randomly selected the remaining sample(s) after two traversals would be heuristically appropriate.

An individual can not be selected more than once during the fractional phase, since the selected individual's expected value is set to zero. Hence this remainder sampling algorithm has Minimum Spread. However, some bias is present in **RSIS**. This bias occurs if a second population traversal is necessary and is similar to that found in latter stages of a spinning wheel algorithm without replacement. However, the overall bias of **RSIS** will be much less since, typically, most of the samples are obtained in the zero biased, first traversal. Note that "positional bias" may occur if the individuals are ordered. However, in a remainder sampling algorithm, the population is shuffled to prevent excessive cloning. Hence, this potential bias does not exist.

The selection phase of **GAs** requires some sequential processing, both in obtaining and sampling the expected values. However, the **GAs**' evaluation, recombination and mutation phases can be executed in full parallel. Hence, increases in the parallel nature of the selection phase may eventually prove useful. **RSIS** is the only acceptable sampling algorithm which can be partially executed in parallel. The fractional phase can be performed in full parallel but must precede the **O(N)** sequential, integral phase.

"C" Code Fragment for Partially Parallel **RSIS**

```
/* sample fractions in full parallel */
/* copy population and mark selected ind.s */
/* code of jth processor */
parbegin
  NextPop.Ind[j] = CurrPop.Ind[j];
  NextPop.flag = (ExpValFract[j] > Rand());
parend;

/* sample integers sequentially */
/* overwrite unmarked individuals first */
k = 0;              /* pointer to NextPop */
set = 1; /* availability flag */
for (i=0; i<N; i++)
   for ( ; ExpVal[i] >= 1.0; ExpVal[i]--)
   {  while ( NextPop.flag[k] == set )
         if (++k == N)  k = set = 0;
      NextPop.Ind[k++] = CurrPop.Ind[i];
   }
```

In the parallel fractional phase, the current population is copied into the next population and the selected individuals are marked with a 1. This mark indicates the positions' unavailability. During the integral phase, the unmarked positions will be filled first. Two potential problem conditions exist: the fractional phase may select too few or too many individuals. If too few are selected, then some unmarked individuals will not be overwritten during the integral phase and hence will be considered selected. This is equivalent to a standard **RSIS** implementation employing random selection on the second traversal. (If this proves undesirable, a sequential, stochastic, second traversal could easily be performed.) If fractional selection choses too many individuals, then integral selection will require additional positions. However, ⌊expected value⌋ must be guaranteed to maintain Minimum Spread. Thus, some individuals selected during the fractional phase must be replaced. This replacement does not alter the performance characteristics of **RSIS**, since sequential **RSIS** would not have chosen these extra individuals in the first place. In either case, Minimum Spread is maintained and the low bias of **RSIS** remains uneffected.

4. Stochastic Universal Sampling

"Stochastic Universal Sampling" (**SUS**) is a simple, single phase, **O(N)** sampling algorithm. It is zero biased, has Minimum Spread and will achieve all **N** samples in a single traversal. However, the algorithm is strictly sequential.

"C" Code Fragment for **SUS**

```
ptr = Rand();
for ( sum=i=0; i<N; i++)
   for (sum += ExpVal[i]; sum > ptr; ptr++)
      SelectInd(i);
```

On a standard spinning wheel, there is a single pointer which indicates the "winner". **SUS** is analogous to a spinning wheel with N equally spaced pointers. Hence, a single spin results in **N** "winners". Since the sum of the population's expected values = **N**, the pointers are exactly 1.0 apart. Thus an individual is guaranteed ⌊expected value⌋ in samples and no more than ⌈expected value⌉. Hence, **SUS** has Minimum Spread. Furthermore, in a randomly ordered population, an individual's selection probability is based solely on the initial spin and the magnitude of his expected value. Hence, **SUS** has zero bias. Like remainder sampling algorithms, the population must be shuffled before crossover. Therefore, positional bias can not exist in **SUS**, either. (Note that, in general, any number of samples, **n**, can be obtained by letting **ptr** range within [0..**N**/**n**) and incrementing it by **N**/**n** after

each selection.) In a sequential environment, SUS seems to be the optimal sampling algorithm. In the next section, an empirical analysis of RSIS and SUS is presented.

5. Empirical Analysis

This analysis investigates the severity, direction and progression of bias in RSSwoR, sequential RSIS and SUS. The severity is important from an implementation standpoint. Although bias can be proven theoretically[2], it is of less interest if it does not alter performance. The direction of the bias indicates which individuals are favored and the progression indicates when the bias occurs. Since we are concerned only with the sampling algorithm, a full execution of GAs is not necessary. Rather, we simply maintain a population of expected values. This allows a single expected value distribution to be sampled, repeatedly. We use linear distributions specified by the parameter MAX. Given $1.0 \leq MAX \leq 2.0$, the expected values' range is (2−MAX..MAX), see Figure 1.

Bias Severity

The severity of the bias is given by its average actual affect on selection, ie. by the number of individuals not selected that should be, or visa versa. The "Fertility Factor", (FF), is defined to be the percentage of the population selected to reproduce[1]. The FF indicates the number of distinct individuals that are selected. Therefore, the difference between the experimentally observed FF and the theoretically expected FF[2] indicates the severity of the bias. In Experiment 1, the sampling algorithms are executed 1,000,000 times for each of ten MAX values between 1.1 and 2.0. The average FFs of each sampling algorithm are compared in Figure 2. Figure 3 presents the bias severity, the difference between the experimental and theoretical FFs.

Clearly RSSwoR is severely biased. It has been empirically shown that in GAs, expected values seldom exceed 1.2[1]. Yet for such a case (MAX = 1.1) approximately 3% of the total population is erroneously not selected. Thus, for N=100, an average of 3 individuals are lost due to bias during each generation. The bias in RSIS is much less severe. For N=100, RSIS averages losing only 1 individual due to bias in every 3 generations. This is an order of magnitude better than RSSwoR. The zero bias expected for SUS is empirically supported by this experiment. Note that, Experiment 1 simply indicates favoritism between two groups: those selected in the integral phase versus those not selected in the integral phase. Hence, some decline in the bias severity as MAX→2.0 is due to the increasing similarity between these group's fractions, see Figure 3.

Bias Direction

We define an individual's "Bias Factor" as the ratio of his actual sampling probability to his expected value. By comparing Bias Factors, we can characterize the direction of the bias, ie. what type of individuals are favored? Since bias only occurs in the fractional phase and the integral phase is deterministic, the Bias Factors will be based solely on the fractional expected values. In Experiment 2, each sampling algorithm is executed 10,000,000 times with MAX = 1.1. This MAX value is chosen to closely resemble a typical expected value distribution. The actual sampling frequencies of each fractional expected value are obtained and used to calculate the Bias Factors plotted in Figure 4. Note that for MAX = 1.1 the fractional expected values form a split range (0 .. 0.1) and (0.9 .. 1.0).

RSSwoR and RSIS favor smaller fractions. Furthermore, the Bias Factor monotonically decreases with increasing fractional magnitude. Both of these results can be proven theoretically[2]. Experiment 1's indication of the sampling algorithms' relative bias is also confirmed, as is its affect on the FF. (In this example, individuals with small fractions have expected values in the range (1.0 .. 1.1). Hence, they are selected during the integral phase. A favorable bias toward these individuals will decrease the FF, see Figure 1. Furthermore, as MAX increases, these fractions increase, thereby decreasing the bias severity, see Figure 3.) Experiment 2 indicates that SUS does not favor any individuals and hence, is empirically zero biased.

Bias Progression

Is the bias constant throughout the fractional selection phase or does it vary? This is important both in understanding the bias and for systems where the generation gap, (fraction of the population replaced during each generation), is less than 1.0. To determine the progression, we must know the sampling probability each individual experiences whenever each sample is taken. This is obtained by recording *when* each selected individual is sampled. Long term averages reveal Bias Factors for each selection cycle, ie. the Bias Factor each individual experiences whenever the 1^{st} selection is made, whenever the 2^{nd} selection is made, etc. In Experiment 3, the sampling algorithms are executed 10,000,000 times with MAX = 1.1 and the Bias Factors are determined. Due to

the unwieldy amount of data, only two arbitrarily chosen, representative individuals are compared: the fifth smallest fraction (0.009) and the fifth largest fraction (0.991). Figure 5 presents these individuals' Bias Factors, though some stochastic fluctuation is still evident.

Clearly, the bias in **RSSwoR** gets progressively worse. This is intuitive – as more fractions are removed from the "spinning wheel", the bias increases. For **RSIS** the zero bias of its first population traversal is evident in Figure 5. The second population traversal has constant bias, however it depends upon the number and nature of the first traversal's samples. Hence the apparent steady, late increase in **RSIS** bias is misleading. It is a direct result of the variable length of the zero biased, first traversal of the population. As a consequence of the increase during latter selections for both **RSSwoR** and **RSIS**, the amount of overall bias is strongly dependent upon the generation gap. This dependence upon the generation gap may in fact have affected previous determinations of optimal generation gap settings[4,6]. Due to the simultaneity of **SUS** – all **N** samples are simultaneously determined by the single random number generated – the concept of a sample cycle is inapplicable.

6. Conclusions

The commonly used sampling algorithm **RSSwoR** is severely biased and is **O(NlogN)**. Whereas efficiency is always desirable, accurate performance (zero bias) is crucial. Hence, two new sampling algorithms are presented. The first algorithm, **RSIS**, has Minimum Spread, optimal **O(N)**, and an order of magnitude less bias than **RSSwoR**. Furthermore, it can be partially implemented in parallel. The second algorithm, **SUS**, has Minimum Spread, optimal **O(N)**, and zero bias. Yet it can only be implemented sequentially.

Empirical evidence is presented that reveals the relative bias of these three sampling algorithms. The Bias Factors are found to decrease monotonically with increasing expected values. The Bias Factors in **RSSwoR** are also found to grow steadily worse as an increasing number of selections are made. For **RSIS**, zero bias is maintained throughout the algorithm's first traversal, yet increases sharply for later traversals. This results in bias occurring only for the last few selections. **SUS** is shown to be zero biased for all individuals.

The **SUS** algorithm is an optimal sequential sampling algorithm. Its use enables **GAs**, for the first time, to assign offspring according to the theoretical specifications. How much the bias has affected the **GAs**' ability to perform, is unknown. Since the bias reduces the actual **FF**, its removal will impede convergence. Thus, the bias should improve performance on simple, unimodal functions and worsen performance on others. This has not been empirically verified, but will be thoroughly analyzed in [2]. Since simple functions are not the domain for which **GAs** are most useful, a degradation in their performance is not of great concern. Hence, in sequential environments, **GAs** should employ the **SUS** algorithm. It will sample performance on more difficult functions. If a parallel environment is available, **RSIS** may prove valuable, especially if a generation gap < 1.0 is desired.

Acknowledgements

I would like to thank Dr. John J. Grefenstette for directing me toward this research area and for numerous discussions of **GAs** in general. Furthermore, I would like to thank Dr. J. Michael Fitzpatrick for his substantial assistance on the theoretical proofs of bias in **RSSwoR**.

References

1. J. E. Baker, *Adaptive selection methods for genetic algorithms,* Proc. International Conf. on Genetic Algorithms, Ed. J. J. Grefenstette, Carnegie-Mellon University, Pittsburgh, PA, pp.101-111 (July 1985).
2. J. E. Baker, *Balancing diversity and convergence in genetic search,* Ph.D. Thesis, Computer Science Dept., Vanderbilt University, Nashville, TN, (September 1987), (expected).
3. A. Brindle, *Genetic algorithms for function optimization,* Ph.D. Thesis, Univ. of Alberta, Alberta (1981).
4. K. A. DeJong, *Analysis of the Behavior of a class of genetic adaptive systems,* Ph.D. Thesis, Dept. Computer and Communication Sciences, Univ. of Michigan (1975).
5. J. J. Grefenstette, *A user's guide to GENESIS,* Tech. Report CS-84-11, Computer Science Dept., Vanderbilt Univ., Nashville, TN, (August 1984).
6. J. J. Grefenstette, *Optimization of control parameters for genetic algorithms,* IEEE Transactions of Systems, Man, and Cybernetics, SMC-16(1), pp.122-128 (1985).

Sampling Method	Bias	Spread	Cost	Parallelability
Stochastic with Replacement	zero	unlimited $0 \rightarrow N$	O(NlogN)	None
Stochastic with Partial Replacement	medium	upper bounded $0 \rightarrow \lceil ev \rceil$	O(NlogN)	None
Remainder Stochastic with Replacement	zero	lower bounded $\lfloor ev \rfloor \rightarrow \lfloor ev \rfloor + \mathbb{R}$	O(NlogN)	None
Remainder Stochastic without Replacement	medium	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	O(NlogN)	None
Deterministic	high	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	O(NlogN)	None
Remainder Stochastic Independant Sampling	low	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	O(N)	Fractional Phase
Stochastic Universal Sampling	zero	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	O(N)	None

Table 1 -- Comparison of Sampling Methods

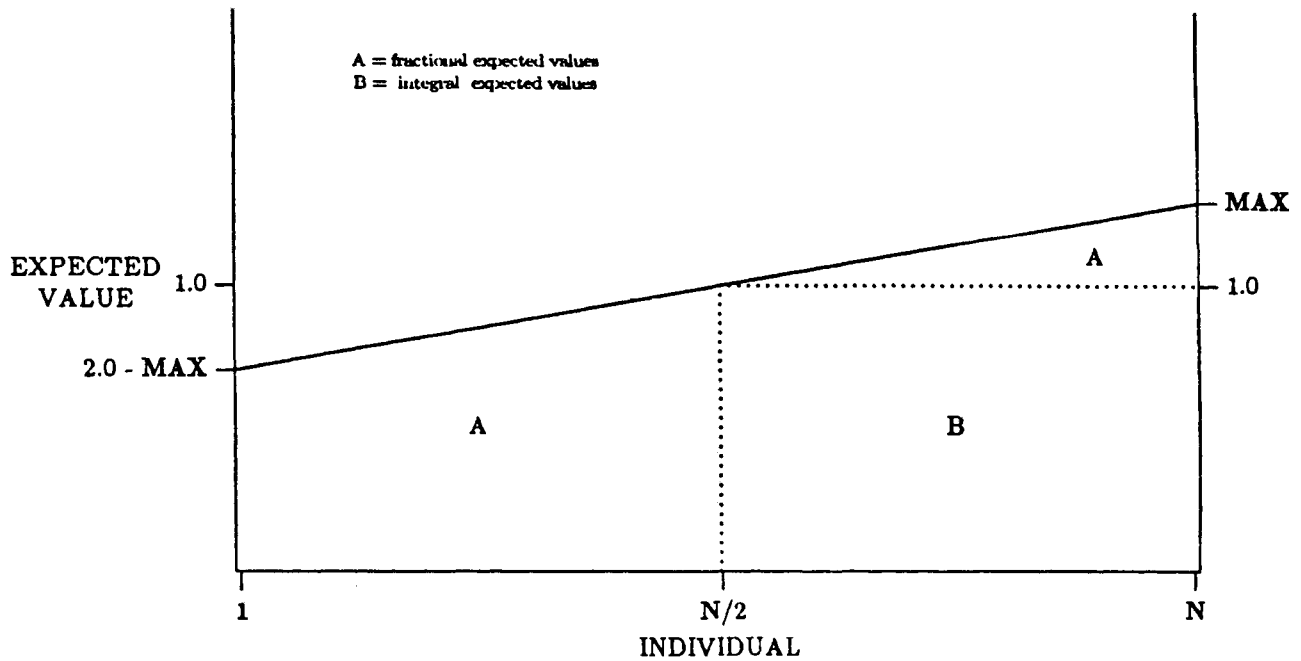


Figure 1 -- Expected Value Distribution

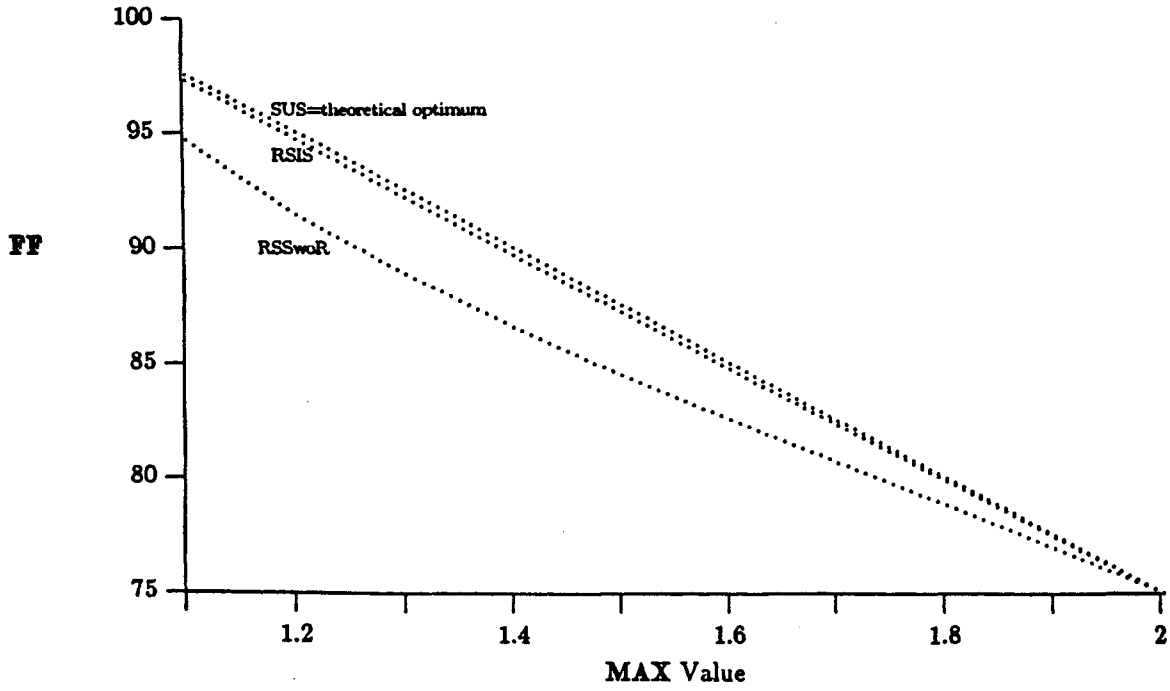


Figure 2 – Observed FFs

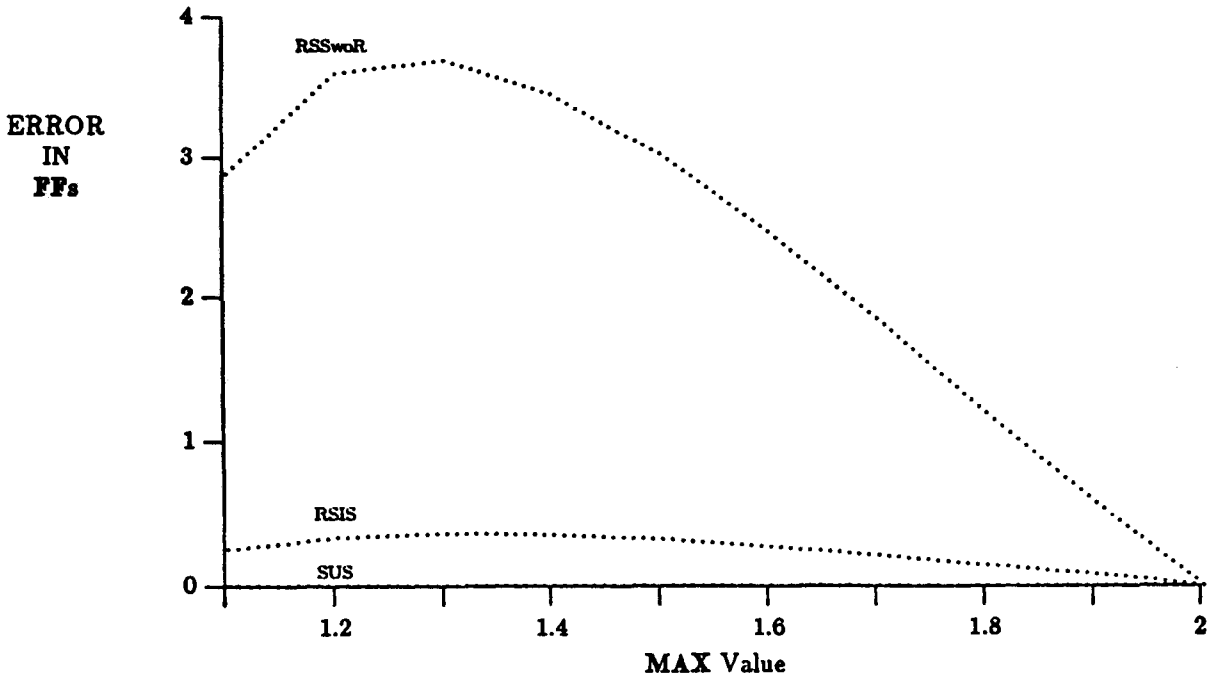


Figure 3 – Bias Severity

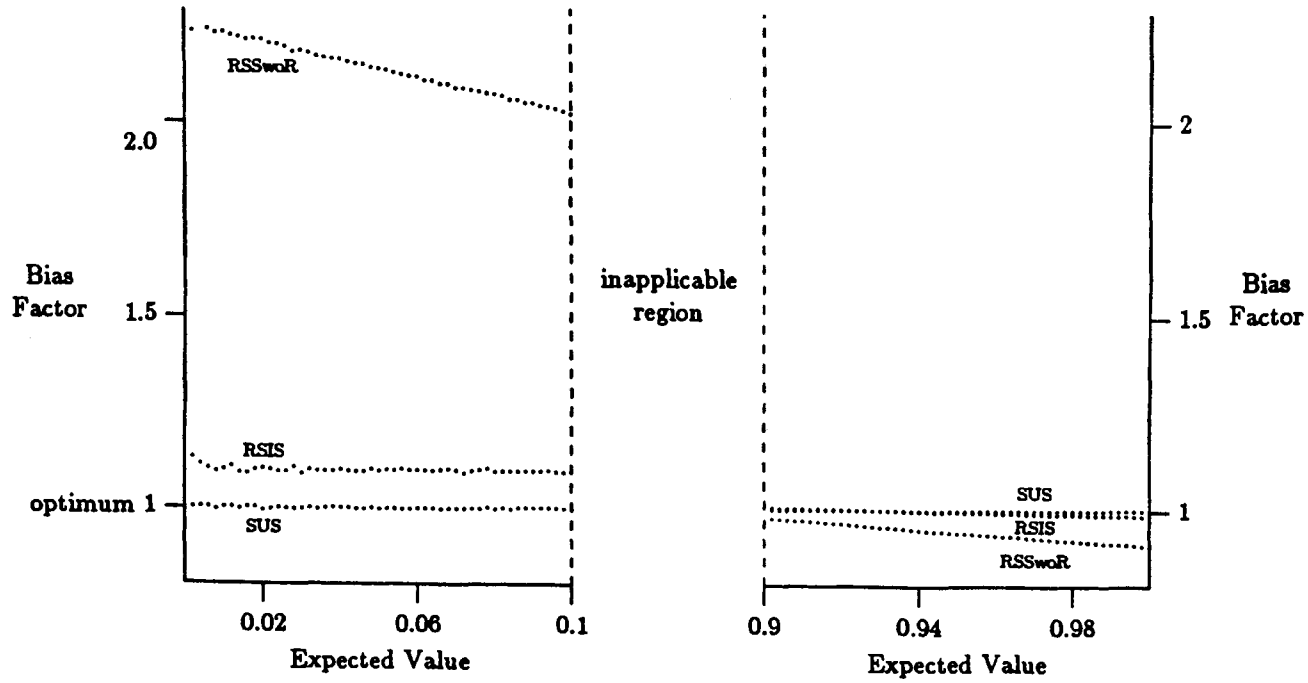


Figure 4 – Bias Direction
(Optimal Bias Factor = 1.0)

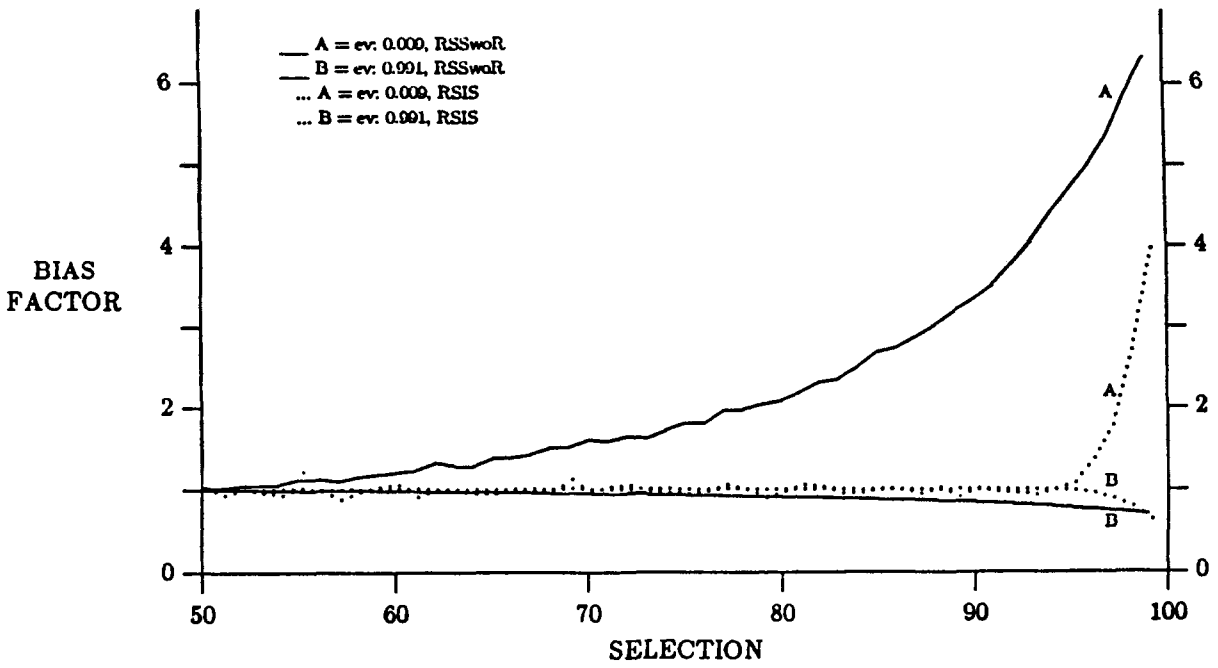


Figure 5 – Bias Progression
(Optimal Bais Factor = 1.0)

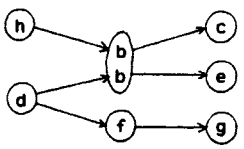
ALTRUISM IN THE BUCKET BRIGADE

Thomas H. Westerdale
Birkbeck College, University of London

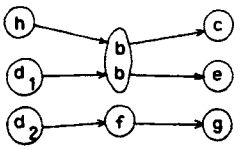
Abstract --We examine the bucket brigade in a simple illustrative production system. We employ a penalty scheme to produce an interpretable bucket brigade reward scheme. The result is a reward scheme that looks like a genetic scheme with altruism added.

1. INTRODUCTION: A PRODUCTION SYSTEM VIEWED AS A RAILWAY

We shall investigate the action of certain reward schemes in a simple illustrative production system and environment. The system consists of seven productions: h, b, c, d, e, f, and g. I call the set of productions that can fire in the current time unit the eligibility set. It is of course a function of environment state. Our eligibility situation can be summarized with a simple diagram -- see Fig. 1.



System and Environment
Fig. 1



With Station d Split
Fig. 2

After each production fires, the eligibility set is the set of those productions it is connected to by arrows. The possible firing sequences are the left to right paths in the figure.

The figure looks like a railway map, where the productions are stations. Trains on this railway always move from west to east. The stations can be seen to form three groups: the western, central, and eastern group. The railway layout is on a cylindrical planet and so trains leaving eastern stations proceeding eastwards, pass around the far side of the planet and appear on the western edge of the map. Their next stop will be one of the western stations, h or d. In other words, emanating from each eastern station there are two arrows (not shown) pointing to h and d. Station b has two platforms. Trains arriving from h arrive at the northern platform and they always depart for c. Trains arriving from d arrive at the southern platform and they always depart for e.

On a trip around the planet, a train takes one of three routes: the northern, hbc; the middle, dbe; or the southern, dfg. A sequence of productions is a permissible firing sequence just if that sequence is a sequence of successive stations on a possible railway journey.

With every train that arrives at station c, the railway company is paid payoff $\mu(c)$, whereas with every train that arrives at station e the company is paid $\mu(e)$, and with every train that arrives at station g the company is paid $\mu(g)$. These three fares are fixed and unchanging. Arrivals at western and central stations yield no such income, no payoff.

Attached to each station is a positive real number called the station's availability. When a train has a decision to make as to which track to take (as for example when it leaves station d) then it makes a probabilistic choice from among the stations in the eligibility set, chosing with probabilities proportional to the stations' availabilities. (Systems in which the driver's choice is deterministic [1] are more common in practice, but harder to analyze. We shall not discuss them, but see section 7 for a comment on their rationale and analysis.) It will be the job of the reward scheme to adjust the availabilities in an attempt to maximise company income per unit time. For example, if $\mu(g)$ is very high then the reward scheme will want to have a high availability of f.

When the reward scheme increases a station's availability we say the station has been rewarded. The amount of the increase is the station's reward.

2. TWO RAILWAY REWARD SCHEMES

Our simple system and environment has a periodicity of 3. Reward schemes discussed in this paper will be specifically designed to handle systems and environments which have a periodicity of 3. We have discussed elsewhere [2,3] the handling of more general systems and environments.

> Scheme A: Each train keeps a record of the stations it passes through while crossing the map from west to east. When the train arrives at an eastern station, the payoff received is added to the availabilities of the last three stations the train has passed through.

Scheme A is not a subgoal reward scheme, so its analysis is comparatively easy. (I said "comparatively". See the analysis in [2].) Bucket brigade schemes, on the other hand, are subgoal reward schemes. Let us move directly to a bucket brigade scheme. In this scheme each station holds a certain amount of cash, called its cash balance. Every time a train arrives at a station, that station sends a hundredth of its cash to the station from which the train arrived. Thus cash moves east to west around the planet, carried by the "bucket brigade".

But the cash management at eastern stations is rather different. Eastern stations throw away the cash they receive from the bucket brigade.

Instead, when a train arrives at an eastern station x, the cash balance of x is incremented by the payoff $\mu(x)$.

Scheme B simply sets the availability of each station equal to its cash balance.

In Scheme A, trains coming from h end up at c, and their payoff, $\mu(c)$, rewards station h, but does not reward station d. In Scheme B that payoff rewards d as well. In other words, cash traveling through station b can change platforms, whereas trains cannot.

Under the bucket brigade, the cash balances of the stations will fluctuate around an "equilibrium value", at which the expectation of cash inflow per train arrival is equal to the expectation of the cash outflow. For any station x, we write $\tilde{x}$ to mean the equilibrium value of the cash balance of x. Thus we have $\tilde{c} = 100 * \mu(c)$, $\tilde{e} = 100 * \mu(e)$, $\tilde{g} = 100 * \mu(g)$, and $\tilde{f} = \tilde{g}$.

For analysis purposes, it is sometimes useful to modify the bucket brigade as follows: All payoffs are multiplied by 100, and when a train arrives at a station, that station sends its whole cash balance (rather than only a hundredth) to the station from which the train arrived. This modification produces what we will call the modified bucket brigade. The equilibrium cash balances are unchanged by the modification.

Scheme B looks interesting, but it is never used and it is clearly wrong. In fact it is clearly wrong in even simpler systems and environments. Suppose for example that there is no northern route on the map, that there are no stations h and c and that station b has only one platform. Then we have both $\tilde{f} = \tilde{g}$ and $\tilde{b} = \tilde{e}$. Suppose that $\mu(g) = 2 * \mu(e)$, so that $\tilde{f} = 2\tilde{b}$. Then the trains leaving d take the route to f about twice as often as the route to b. As they do this, more and more evidence piles up that the f route is the better route, but instead of sending more and more trains to f, the system keeps sending about a third of its trains to b.

3. A MORE BIOLOGICAL BUCKET BRIGADE

In biologically motivated systems one generally sets the reward proportional to payoff. That is, the change in availability is proportional to payoff. This is the case in Scheme A. But in Scheme B, the cash balances are proportional to payoff, so the cash balances ought not to be the availabilities, but rather ought to be the change in the availabilities. So we have

> Scheme C: Each station keeps two numbers, its cash balance and its availability. The cash balances are maintained exactly as in Scheme B. The availabilities are updated as follows: Every time a train arrives at a station, that station's availability has added to it the product of K and the station's cash balance. (K is a small positive constant.)

It is Scheme C, not Scheme B, which we should have been comparing to Scheme A. If we use the modified bucket brigade, then Schemes C and A are almost identical. If we look at the east to west path followed by any penny in the modified bucket brigade, we can see that if the penny could not change platforms at b then Scheme C would be just like Scheme A.

Let us look more carefully at what happens in Scheme C. Note that $\tilde{h} = \tilde{b}$ whatever the availabilities. The value of $\tilde{b}$ depends on the availabilities, since these determine the ratio between the number of trains passing through the northern platform of b and the number of trains passing through the southern platform of b.

Let us let the station name without the tilde stand for the availability of that station. It should be clear from context whether it is the station or its availability that is meant.

To obtain a formula for $\tilde{b}$ in terms of availabilities, we need to know the proportion of trains that go through h, and the proportion that go through d. We define $q = h + d$, the total availability of the western stations. We define $H = h/q$ and $D = d/q$. The capital letters can be thought of as normalized availabilities, or under certain circumstances, as probabilities. Similarly for the central stations we let $r = b + f$, $B = b/r$, and $F = f/r$. There are three routes for trains going from west to east, and their probabilities are H for route hbc, DB for route dbe, and DF for route dfg. The first two routes go through b and so it is easy to see that $\tilde{b} = (H\tilde{c} + DB\tilde{e})/(H + DB)$. Similarly, $\tilde{d} = B\tilde{b} + F\tilde{f}$. The $\tilde{b}$ is the equilibrium value of the cash balance of b for the given availabilities. The cash balance will fluctuate around this value $\tilde{b}$ provided the availabilities are held constant.

They will of course be changing, but we assume slowly. Thus we can approximate the behavior of the system by the usual continuous deterministic model in which we think of the trains as fluids and the tracks a pipes, the volume of fluid flowing through a pipe reflecting the average number of trains passing along that track. Under Scheme C the availability of, for example, f increases by $K\tilde{f}$ (on average) every time a train enters f. Thus f', the rate of change of the availability of f, is $K\tilde{f}$ times the rate at which trains enter f. If n trains leave the western stations per hour on average, then there are on average nDF of these passing through f and so $f' = K\tilde{f}nDF$, in our continuous model. By the same reasoning, there are $nH + nDB$ trains passing through b on average, so $b' = K\tilde{b}n(H + DB)$. We can similarly calculate h' and d'. Then, defining $\tilde{q} = H\tilde{h} + D\tilde{d}$ and $\tilde{r} = B\tilde{b} + F\tilde{f}$, and using ordinary differential calculus, we can obtain the other rates in Table 1.

So what does Table 1 tell us about Scheme C? Not a lot, I'm afraid. Perhaps the three routes,

$h' = Kn\bar{H}\bar{h}$	$H' = (1/q)KnHD(\bar{h} - \bar{d})$
$d' = Kn\bar{D}\bar{d}$	$D' = (1/q)KnHD(\bar{d} - \bar{h})$
$b' = Kn(H + DB)\bar{b}$	$B' = (1/r)KnF((H + DB)\bar{b} - DB\bar{f})$
$f' = KnDF\bar{f}$	$F' = (1/r)KnF(DB\bar{f} - (H + DB)\bar{b})$
$q' = Kn\bar{q}$	
$r' = Kn\bar{q}$	

Table 1: Rates for Scheme C

hbc, dbe, and dfg can be thought of as three genotypes with respective fitnesses $\tilde{c}$, $\tilde{e}$, and $\tilde{g}$. There would need to be some fancy recombination rules.

Let's look at the three routes under Scheme C. The proportion of trains on the three routes is H, DB, and DF, respectively. The changes in these three are easily calculated. H' we have already:
$(DF)' = (1/q)KnDF(H\bar{f}-H\bar{b})+(1/r)KnDF(DB\bar{f}-H\bar{G}-DB\bar{b})$; and
$(DB)' = (1/q)KnDB(H\bar{f}-H\bar{b})+(1/r)KnDF(H\bar{b}+DB\bar{b}-DB\bar{f})$.
This doesn't look very helpful.

4. AN ATTEMPT AT STATION SPLITTING

Let's try to go part way toward separating our system into the three routes, part way toward thinking of it in a genetic manner. It looks as if a fork like that at d can be treated as if it were two stations d_1 and d_2, with trains from d_1 going only to b and trains from d_2 going only to f. See Figure 2. A train leaving an eastern station goes to h, d_1, or d_2 according to the 3 availabilities of those stations.

The hope is that Fig. 2 will behave just as Fig. 1 did if $d_1 = dB$ and $d_2 = dF$. Unfortunately, this doesn't work. $D_1' = (1/q)KnDFDB(\bar{b} - \bar{f})$, if we represent it in Fig. 1 quantities, and $D_2' = (1/q)KnDF(H + DB)(\bar{f} - \bar{b})$. If our splitting of station d really results in an equivalent system, then D_1' should equal our Fig. 1 $(DB)'$, and D_2' should equal our Fig. 1 $(DF)'$. Even when $r = q$ these quantities are not equal. When $r = q$ we have
$(DF)' = (1/q)KnDF((H + DB)(\bar{f}-\bar{b}) - H\bar{b})$,
$(DB)' = (1/q)KnDFDB(\bar{b}-\bar{f}) + (1/q)KnDH(B\bar{f} + (F-B)\bar{b})$,
and $D_2' - (DF)' = (1/q)KnDFH\bar{b}$.

5. THE PENALTY SCHEME

Looking at $D_2' - (DF)'$, we see that the Fig. 1 dfg route seems to be getting less reward than it should. It looks as if cash from b has something to do with the discrepancy. If we look at b' and f' in Table 1, we can see what might be causing the problem. b' is too big. The availabilities b and f are used by trains leaving d. Under Scheme C, all those irrelevant trains leaving h proceed on to b and boost its availability. Clearly the H in the formula for b' does not belong.

This is part of a more general problem. A sensible reward scheme penalizes a production whenever it is a member of the eligibility set. Otherwise, there is no incentive for a production to take itself out of the eligibility set when there is little likelihood of its doing good. I have proposed, in a non-bucket-brigade context, that when a production is rewarded, its reward is then subtracted from all the members of the eligibility set in proportion to their availabilities [2]. If we do this we obtain the rates given in Table 2.

$h' = KnHD(\bar{h} - \bar{d})$	$H' = (1/q)KnHD(\bar{h} - \bar{d})$
$d' = KnHD(\bar{d} - \bar{h})$	$D' = (1/q)KnHD(\bar{d} - \bar{h})$
$b' = KnDBF(\bar{b} - \bar{f})$	$B' = (1/r)KnDBF(\bar{b} - \bar{f})$
$f' = KnDBF(\bar{f} - \bar{b})$	$F' = (1/r)KnDBF(\bar{f} - \bar{b})$
$q' = 0$	
$r' = 0$	

Table 2: Rates for Scheme C with Penalties

We can now go ahead and derive as before
$(DB)' = KnDF((1/q)HB(\bar{f} - \bar{b}) + (1/r)DB(\bar{b} - \bar{f}))$, and
$(DF)' = KnDF((1/q)HF(\bar{f} - \bar{b}) + (1/r)DB(\bar{f} - \bar{b}))$.
Splitting station d and using the penalty scheme gives
$D_1' = (1/q)KnDBDF(\bar{b} - \bar{f})$ and
$D_2' = (1/q)KnDF(1 - DF)(\bar{f} - \bar{b})$.

It is now easy to show that to have $(DB)' = D_1'$ and $(DF)' = D_2'$ it is necessary and sufficient that $r = d$. The penalty scheme was developed for a non-subgoal reward scheme like Scheme A, but is applied here to a bucket brigade. And its application suddenly makes sense of the equations: The availabilities b and f are for the use of trains leaving d. One will be able to split d just if the availabilities of b and f add up to the availability of d. I questioned [4] whether the penalty scheme made sense in the bucket brigade context. Evidently it does.

On the other hand, the penalty scheme doesn't really allow station splitting. With the penalty scheme, r is constant, so keeping $d = r$ means d can't be rewarded. Since q is also constant, that means h can't be rewarded either. That's no good.

6. RESTORING THE PENALTY

The penalty scheme was developed [2] as a practical equivalent to a biologically motivated scheme which instead of penalizing productions inside the eligibility set, rewards productions outside it. The biologically motivated scheme can be thought of as identical to the penalty scheme except that after penalizing, the penalty is restored as follows: All the stations of the current locus (the geographical region at which the train is arriving: west, central, or east) are rewarded by the total penalty in proportion to their availability. The penalized eligibility set is a subset of the locus, so for the locus as a whole, the penalty is restored.

There are two obvious ways of implementing this scheme. One is by making the reward, then penalizing across the eligibility set, and then distributing the restoration across the whole locus. Another more suggestive, but much noisier implementation is as follows. When a train arrives

at a station, calculate the reward (K times the station's cash balance), but don't give it to that deserving station necessarily. Choose an arbitrary station from the same locus (choose with probabilities proportional to availabilities). If the chosen station is outside the eligibility set, give the reward to the chosen station. If the chosen station is inside the eligibility set, give the reward instead to the deserving station.

Table 3 gives the rates for Scheme C with restored penalties.

$h' = KnH\tilde{h}$	$H' = (1/q)KnHD(\tilde{h} - \tilde{d})$
$d' = KnD\tilde{d}$	$D' = (1/q)KnHD(\tilde{d} - \tilde{h})$
$b' = KnB\tilde{b}$	$B' = (1/r)KnDF(\tilde{b} - \tilde{f})$
$f' = KnF(H\tilde{b} + D\tilde{f})$	$F' = (1/r)KnDF(\tilde{f} - \tilde{b})$
$q' = Kn\tilde{q}$	
$r' = Kn\tilde{q}$	

Table 3: Rates for Scheme C with Restored Penalties

If we use Table 3 to calculate (DB)' and (DF)', and if we ask what is necessary for these to equal D_1' and D_2' respectively after station splitting, we come up with the ridiculous requirement that D = 1. Why should this be so? Can we make any sense of Table 3?

7. ALTRUISM

If we set r = q, then Table 3 makes sense. We can think of a population of size q * q, consisting of individuals of the four genotypes (see Table 4). The number of individuals in the population of genotype hb is simply the product hb, and the same for the other genotypes. So the population is in Hardy-Weinberg equilibrium.

We can view the operation of the scheme as follows. Each time unit, n genotypes are chosen at random from the population. Each of these genotypes is then run as follows. It is given to a train driver who runs his train west to east, using the genotype as a prescription to tell him which track to take when the tracks fork. When the train arrives at the eastern station, the payoff (μ(c), μ(e), or μ(g)) is passed to the modified bucket brigade, which runs it back east to west. Table 4 shows the routes back west that the payoff can take, and for each route, Table 4 shows which stations can be rewarded.

The top line of the table says that when hb is the genotype, the payoff will be $\tilde{c}$ (since the train will go by the northern route). The payoff has two possible routes back west, via b and h or via b and d. (Remember, money can change platforms at b.) Whether it goes from b to h or from b to d depends on where the next train entering b comes from. The probability that it comes from h is clearly H/(H + DB). We can see that if it follows the route via b and h, it has a chance to make one reward to a central station (when it is at b and a train arrives at b) and one to a western station (when it is at h and a train arrives at h). Now in

geno-type	Pr	payoff	routes of payoff	Pr	rewards to	Pr
hb	HB	$\tilde{c}$	hb	H/(H + DB)	hb	B
					hf	F
			db	(DB)/(H + DB)	db	1
hf	HF	$\tilde{c}$	hb	H/(H + DB)	hb	B
					hf	F
			db	(DB)/(H + DB)	db	1
db	DB	$\tilde{e}$	hb	H/(H + DB)	hb	B
					hf	F
			db	(DB)/(H + DB)	db	1
df	DF	$\tilde{g}$	df	1	df	1

Pr = probability

Table 4: Payoff Routes and Rewards

rewarding the central station, b will be the deserving station, whereas the chosen station will be b with probability B, and f with probability F. The eligibility set is {b}, so if f is chosen the reward goes to the chosen station f, whereas if b is chosen the reward goes to the deserving station b. In rewarding the western station, h will be the deserving station, whereas the chosen station will be h with probability H and d with probability D. The eligibility set is {h,d}, so whether h or d are chosen, the reward will in any case go to the deserving station h. Thus if the payoff follows the route via b and h, the reward will go to h and b with probability B, and to h and f with probability F. This is summarized in the first two entries in each of the last two columns of the table.

Table 4 can be used to calculate the rates. In each time unit, n trains run from west to east, so for example to get h' we need only run down the second to the last column, choosing all entries containing an h. For each of these entries, we multiply together K, n, the reward in column 3, and the three probability columns. We then sum the results. The rates we so obtain are those given in Table 3.

There is a certain strange uniformity in Table 4. The last four columns have identical entries for the genotypes hb, hf, and db. In other words, these three genotypes distribute their reward identically. So we can simplify Table 4 to Table 5.

genotype	payoff	rewards to	probability
hb	$\tilde{c}$	hb	(hb)/(hb + hf + db)
hf	$\tilde{c}$	hf	(hf)/(hb + hf + db)
db	$\tilde{e}$	db	(db)/(hb + hf + db)
df	$\tilde{g}$	df	1

Table 5: Reward Distribution

The last column of Table 5 is the product of columns five and seven of Table 4. Table 5 tells us that when hb is the genotype, the payoff is $\tilde{c}$ and this reward is given to one of the genotypes hb, hf, or db, according to the probabilities in the last column.

So we can think of the four genotypes in our

population as being grouped into two nations. One nation consists of all those individuals whose genotypes are hb, hf, or db. The other nation consists of all those whose genotypes are df. Each genotype has a value (Table 5 column 2). If the scheme were a straightforward genetic scheme, the number of children per time unit of each individual would be K times that individual's value, and then complete recombination would take place. In this scheme, however, each individual altruistically hands his payoff to an arbitrary individual which he selects at random from his own nation. Thus the payoff determines the number of children of the arbitrary recipient rather than of the donor. Of course the payoff does at least remain within the same nation. So our system can indeed be divided into two parts, but evidently not in the way we intended.

The system seems to exhibit genuine altruism and not simply kin selection [5]. Each genotype would do better for its genes if it didn't give away its payoff. Group selection is taking place, the nations being the groups.

Suppose we remove from the scheme the altruistic handing over of payoff, giving us a straightforward genetic scheme. If we had begun not with Scheme C, but with Scheme A, and if we had added the penalties and then the restorations to Scheme A, then the scheme we would have ended up with would have been that straightforward genetic scheme. Thus it is the altruism that distinguishes Scheme C from Scheme A. It is altruism when the pennies change platforms at station b.

To what extent can the previous analysis be performed in a more complex system and environment? The short answer is that I don't know. Elimination of the periodicity does not seem to me to present the more serious problem. It is the introduction of fancier railway branching that looks the more worrying. Suppose for example that there were an additional route hjk in the extreme arctic. Then how could alleles specify choices both for drivers leaving h and for those leaving d? I suggested in [2] that to handle more complicated branchings, the alleles should be orderings of the stations at a given locus. That trick might work.

In many published systems [1] the train driver makes a deterministic rather than a probabilistic choice when the track forks. The rationale is that since a higher cash balance will ultimately lead to a very much higher availability, we can forget about availabilities and ask the driver to always head for the station with the higher cash balance. Such a system relies on noise to shake the system out of non-maximal equilibria. Analysis is difficult because results depend on assumptions regarding noise. Such an analysis is not attempted here.

8. CONCLUSIONS

By a continual process of modifying our bucket brigade scheme and trying to make sense of it, we have arrived at a scheme which has a simple description in one simple system and environment. This scheme employs a system of penalties. It looks like a genetic scheme with altruism added.

The sequence of analyses and changes reported here are genuinely in the order I followed, but with the false trails removed. I really did try the Scheme C first with no penalty, and was puzzled by the messy equations. I had previously used the penalty scheme in a non subgoal reward context [2], but I genuinely forgot about it and so was led astray. The nature of my difficulties reminded me of it, and it seemed to work. I had previously [4] wondered whether the penalty scheme made sense in a subgoal reward context. I'm convinced now that it does. If I had used it from the first, I wouldn't be so sure.

If we had tried different modifications we would have ended up with a different scheme, and perhaps if we had been cleverer we could have made sense of some of the schemes we rejected as nonsensical. Still, what we have shown is that we can ask our questions rigorously enough that the answer can (and frequently does) come out "no". That we can be forced by the facts into an explanation different from the one we intended is an indication that we are doing more than merely making qualitative generalizations. But perhaps not very much more.

References

1. J. H. Holland, "Escaping Brittleness: the Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule Based Systems", pp. 593-623 in *Machine Learning II*, ed. T. M. Mitchell, Morgan Kaufmann, Los Altos, Calif. (1986).

2. T. H. Westerdale, "A Reward Scheme for Production Systems with Overlapping Conflict Sets", *IEEE Trans. Syst., Man, Cybern.* Vol. SMC-16(3), pp.369-383 (May/June 1986).

3. T. H. Westerdale, *An Automaton Decomposition for Learning System Environments*, Submitted.

4. T. H. Westerdale, "The Bucket Brigade is not Genetic", in *Proceedings of an International Conference on Genetic Algorithms and their Applications*, ed. John J. Grefenstette, Carnegie-Melon University, Pittsburgh (1985).

5. John Maynard Smith, *The Theory of Evolution*, Penguin, Harmondsworth, Middlesex, England (1966).

SCHEMA RECOMBINATION IN A PATTERN RECOGNITION PROBLEM

Irene Stadnyk *
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87545. †

Abstract

A simple pattern recognition problem is presented, where a population of binary strings evolves under the genetic algorithm as it tries to match and recognize another population of binary strings. Sampling is used in the genetic algorithm to do matching, which determines fitness, and crowding, which allows subpopulations to form that recognize different pattern strings. To understand what subpatterns the recognizer strings chose as the regularities in the patterns and how these are organized as recognition improves, the theory of schemata is applied to the resulting recognizer string population. The schemata that are formed by recombining other schemata and mutating them with a low level of noise emerge in a default hierarchy. This hierarchy has general properties that apply to any knowledge representation used in such a pattern recognition system.

1 Introduction

Adaptive systems require a dynamic memory in which to store information about the environment. The memory must be organized in such a way so that relevant information can be retreived quickly and can be used to predict states of the environment based on the current states. Feedback from the environment should be used to have the system learn: the memory should be updated using the feedback so that it can make more accurate predictions in the future, even with partial or noisy information about the environment. The memory's predictions are also used to determine the performance or behavior of the system.

The adaptive system builds its memory by trying to find regularities in the environment that occur in more than one environmental state or occur frequently in the environment. These regularities based on similarity and repeateability are then used to form equivalence classes of environmental states that are then used to organize the memory. These form the basis of the learned model of the environment stored on memory.

In a population of adaptive systems, another requirement can be placed on the memory. The memory must be capable of being transmitted from one system to another. That is, a code representing instructions on how to build a functioning memory must be transmitted to the another system. The code consists of discrete units, which are transmitted in some partial sequential order. Once the code is formed, it does not change but rather is subjected to pressures of selection. It is transmitted throughout the population based on its fitness. It is not clear how the transmitted code is decoded and how it affects the structure and information represented in the receiving memory.

Adaptive system memories have been described in the paradigm of adaptive networks. Adaptive networks include autocatalytic networks of chemical reactions [3], the idiotypic network in the immune system [10], classifiers under the Bucket Brigade Algorithm [9], and nets abstracted from these systems [4]. Transmittable adaptive memories include self-reproducing "organsims" in cellular automata [12] and genetic algorithms [8].

This paper presents a model of an adaptive system based on genetics, which has a memory consisting of binary bit strings. The transmittability of the memory is not discusses since presumably the binary strings can be transmitted in a sequence of discrete bits. However, the organization of a memory consisting of binary strings is studied as the binary strings evolve under a genetic algorithm. The system with this memory is a simple pattern recognizer, based on pattern recognition in the immune system. The system's memory is a population of recognizer strings which try to recognize many pattern strings in parallel by picking out regularities in the pattern strings. The genetic algorithm generates new recognizer strings according to fitness. The fitness of each recognizer string is determined by how many bits it matches of how many pattern strings. Recognizer strings with above average fitness reproduce, and of those, the ones with higher fitness produce more offspring. Thus, recognizers that contain "subpatterns" or substrings of one or more pattern strings will prolifer-

*Permanent address: EECS Department, The University of Michigan, Ann Arbor, MI 48109.
†Arpanet address: ims@lanl.gov.

ate through the population. These subpatterns are the schemata that the recognizer population contains and that the genetic algorithm implicitly manipulates. Initial runs of this model show that these schemata evolve into a default hierarchy.

2 The Problem

This pattern recognition problem has been motivated by the pattern recognition that occurs in the immune system. In the immune system[5], one population of antibodies (recognizers) tries to recognize any antigens (patterns) that might invade an organism. By recognizing the antigens, the antibodies tag the antigens, which will then be killed and removed by other cells. The antibodies are formed from gene libraries. From experimental data, we see that these initial antbodies undergo a high rate of mutation once an anitgen is found in the organism. This implies that the initial antibodies must contain some initial structure that can simply be mutated to quickly produce an antibody that recognizes the new antigen. Also, this structure must have been transmitted through the gene libraries. What structure might exist in the gene libraries that have evolved to produce antibodies that are good at recognizing antigens?

In order to study what structure might be found in the gene libraries, a model has been built and simulated which represents the antibodies (recognizers) and antigens (patterns) as binary strings of a fixed length. Several constraints have been put on this model which will be used to study the structures that might occur in the gene libraries. The gene libraries are implicitly represented in this model by the antibodies for which they code. The model constraints are evolutionary constraints, pattern recognition constraints, and covering constraints. The evolutionary constraints have been implemented by the genetic algorithm described in the next section.

The pattern recognition problem is defined as follows:

Population A_r of binary strings of length l
recognizes a binary string $x \in$ population A_p
of binary strings of length l,
where $|A_r| > |A_p|$ if and only if
$\exists$ a non-empty $S \subseteq A_r$ such that
$\forall x' \epsilon S$

$$\sum_{i=0}^{l-1}((\neg(x \uparrow x') \gg i)\&1) \geq T,$$

where $\uparrow$ is the XOR operator, $\gg$ is the shift-right operator, & is the bitwise AND operator, and T is an integer threshold. Thus, a population A_r of recognizer strings recognizes a pattern string when there is at least one recognizer string that has the same bit values in at least T bit positions.

T in the immune system has been experimentally found to be between six and eight amino acids, but in this conceptual model T has been set to five bits, though this can easily be changed. Also, in the real immune system the antibodies form three dimensional configurations so that any T or more positions can try to match the antigen. In the current model, this translates to letting any bit position value of the recognizer string match any other bit position value of the pattern string within the limits of the configuration possibilities. Since the configurations of real antibodies are not known yet, this idea has been abstracted to simply allowing any T bit position values to match with the values in the *same* bit positions in the pattern string. Also, two bit values match if they are not only in the same position in the binary string, but also if they are the same values. In the real immune system, matches occur between complementary values rather than identical values. In this simple model, this does not make a difference. However, if the model were to incorporate the fact that antibodies can match other antibodies as well (where self-recognition and loop creation are important), then complementary matching would have to be the rule. However, in the current model, antibodies as recognizers evolve as they try to match the antigens or patterns, regardless of what other recognizers the recognizer string may match.

Another way of looking at the pattern recognition problem is in terms of schemata. Schemata represent subsets of binary strings by picking out certain bit positions that must have a certain bit value, where the rest of the bit positions can have either value[8]. For instance, the string 011 is contained in schemata 0##, #1#, ##1, 01#, 0#1, #11, 011, and ###, where the #-sign indicates that for that position the schema will accept any value. In this model, the defining bits of a schema would represent those bit values that match between a recognizer string and a pattern string.

The theory of how schemata are manipulated under the genetic algorithm can be used to understand how the recognizer strings are evolving. Since each binary string is a member of 2^l schemata, between 2^l and 3^l *schemata* are implicitly tested and recombined by the genetic algorithm each generation or time step as the *strings* get tested and recombined with respect to their fitnesses. The strings reproduce in proportion to their fitness. Schemata can also be thought of as having fitnesses based on the average of the fitnesses of the strings that are members of that schema. Then as Holland has shown[8], the schemata also reproduce in the population in proportion to their fitnesses. Each schema is implicitly manipulated in the strings that are members of that schema, where the strings are explicitly manipulated by

the genetic operators. What this means for the current pattern recognition problem is that if the genetic algorithm finds one schema in the recognizer population with at least T defining bits that match with a pattern, then the recognizer population recognizes that pattern string. Also, as the genetic algorithm manipulates *schemata* at an exponential rate to find useful ones that match patterns in the pattern population, it also finds recognizer *strings* at an exponential rate to match and recognize the patterns.

The covering problem can simply be stated as follows:

Population A_r covers population A_p
if and only if
$\forall x \epsilon A_r \exists$ a non-empty $S \subseteq A_p$
such that
$\forall\ x' \epsilon S$, x' recognizes x.

That is, every pattern string in the pattern population should be recognized by at least one recognizer string.

The runs of this model should at least be able to show that the recognizer population can evolve under the genetic algorithm to recognize a pattern string as well as cover the pattern population. Once parameter settings and reproduction strategies have been found that give this result, the schemata in the evolving recognizer population can be studied to determine how the recognizer population as a whole organizes information about the patterns as it discovers recognizers to recognize the patterns.

3 The Model

The parameter values for the genetic algorithm are set as follows.

The length of the strings is

$$l = 36$$

bits.

To determine the size of the recognizer population, the length of the schemata one expects to find must be taken into account. Based on recognition in the immune system, the model should be able to let schemata with five or six defining bits develop in the recognizer population. To see what schemata we can expect to find in the recognizer population, we can use a corollary of John Holland's Theorem 6.2.3 [8] that

$$\epsilon < \frac{2n}{l}$$

where ϵ is the error due to crossover (which may break an instance of a schema), n is the the number of defining bits in the schemata one is interested in, and l is the string length. Since $l = 36$, the error for a schema of length 1 is < 0.056, $2 < 0.12$, $3 < 0.17$, $4 < 0.23$, $5 < 0.28$, $6 < 0.34$, and so on. In order for the recognizer population to be able to search for schemata with $n = 6$ defining bits, of which there are $2^n = 2^6 = 64$, the recognizer population size should be about 64 as an upper bound. Schemata with more defining bits might still be found. The recognizer population size has been set to

$$recognizerpopulationsize = 50$$

which is still much greater than $2^5 = 32$ and close to $2^6 = 64$, where the error rate is about 0.3, about as high as the genetic algorithm can tolerate. The initial strings in the recognizer population are created at random.

The pattern population size has been chosen to be

$$patternpopulationsize = 1$$

for one set of runs, and

$$patternpopulationsize = 10$$

for another set of runs to see if the recognizer population can recognize just one string as well as cover several pattern strings. The population is initialized with random strings that do not change over time.

The point mutation rate has been set as low background noise,

$$pointmutationrate = 2$$

two point mutations per time step, where the mutated bits are chosen at random.

The crossover rate has been set to

$$crossoverrate = \frac{1}{3} \cdot recognizerpopulationsize$$

so that in each generation $\frac{1}{3}$ of the recognizer strings are chosen as parents, thus producing that many new strings, and replacing that many old strings. The parents are chosen probabilistically with respect to fitness so that the algorithm results in reproduction with emphasis (on fitness)[8]. Schemata with above average fitness have new copies or samples of them generated in the recognizer population in accordance with the theorem that guarentees minimal performance loss.

Several fitness functions have been considered. With fitness equal to the average number of matching bits of the recognizer string to every pattern string, fitness generally does not increase monotonically nor exponentially as claimed should happen with crossover[8]. To encourage the recognizer strings to evolve with as many consecutive bits matching a pattern string as possible so that it is less likely that the schema of matching bits will be broken by crossover, fitness has been chosen to be

$$f(x) = \frac{\sum_{j=1}^{msample} \sum_{i=1}^{\#jmatches} \frac{m_j(i)}{2^{-m_j(i)}(l-m_j(i)+1)}}{msample}$$

where x is a recognizer string, msample is *matchsample*, $m_j(i)$ is the number of consecutive bit matches between the ith set of consecutive bit matches in string x and the jth pattern string in sample s_m, *#jmatches* is the number of consecutive bit match sequences between string x and the jth pattern string, and l is the length of the strings. For example, if x =01101 and s_m = {11100, 00101}, then $m_1(1) = 3$, $m_2(1) = 1$, $m_2(2) = 3$, and

$$f(x) = \frac{3}{2^{-3} \cdot 3} + \frac{1}{2^{-1} \cdot 5} + \frac{3}{2^{-3} \cdot 3} = 16.4.$$

The *matchsample* is the size of the sample s_m of strings chosen from the pattern population. The sample s_m ,rather than all of the strings in the pattern population, is used to compute fitness. Using this sample helps subpopulations emerge in the recognizer population, as in Booker's simulation [1]. Subpopulations are necessary if all of the patterns are to be recognized by the recognizer population.

The combinatorial bias of the probability of finding m matching bits between two strings, $\frac{1}{2}^m(l - m + 1)$, has been removed by dividing the number of matches by the bias to get the unbiased fitness that would make all lengths of matches equally likely to occur when based on fitness. The probability of finding a match between two bits is $\frac{1}{2}$. For m bits, it is $\frac{1}{2}^m$. Since these bits are consecutive, there are $(l-m+1)$ ways that they can be chosen in a string of length l. Therefore, a sequence of m consecutive matching bits occurs with a probability of $\frac{1}{2}^m(l - m + 1)$ in a string of length l. Dividing by this probability is equivalent to putting more emphasis on longer schemata since the dominating term of $2^{m_j(i)}$ increases exponentially with the length of the substring, $m_j(i)$.

Several replacement strategies have been tried for determining which recognizer strings get replaced by the new offspring strings created by crossover. Choosing a string probabilistically with respect to $\frac{1}{fitness}$ eventually gives rise to a homogeneous population consisting of one highly fit string. Choosing a string that most closely matches the new string, a "crowding" strategy, never allows the fitness to go up since the child too often replaces its parent and thus schemata do not increase in proportion to their fitness as they should. A similar strategy, first tried by DeJong [2] and later by Booker [1] where the closest matching string is chosen from a small random *sample* of the recognizer population does not do much better than the first strategy. A strategy which works well is one which chooses a small sample of recognizer strings, s_c, of size *crowdsample* probabilistically with respect to $\frac{1}{fitness}$. Then from this sample, the string which most closely matches the new string is chosen to be replaced. With this strategy, the recognizer population fitness increases and does not yield a homogeneous recognizer population. The *crowdsample* is determined by how many copies of a schema one wants. A sample of size n would on average include recognizer strings in all schemata that are present in $\frac{1}{n}$ of the recognizer population. Thus, n subpopulations are allowed to coexist in the recognizer population since any subpopulation cannot contain more than $\frac{1}{n}$ recognizer strings. Since the replacement strategy uses fitness to determine which strings to replace, it is consistent with the reproduction with emphasis plan. That is, schema with below average fitness will have copies or samples removed so that their sample size decreases and makes room for an increase in the number of above average schema. Thus, this strategy generally replaces the string with the lowest fitness and which is in the same subpopulation as the new string.

The algorithm to find schemata actually finds a small sample of all schemata present in a string population. The sampling is designed to find the more useful schemata (building blocks) that evolve under the genetic algorithm rather than all of the schemata present in a population. The algorithm to find these schemata is:

1. Choose a sample, s_1, of the population in which want to find schemata (e.g. n most fit strings).
2. Find the schemata in each string in sample s_1. For each string in sample s_1:
   (a) Get a random sample, s_2, from the population.
   (b) Check string against each string from s_2 to find which bits match.
   (c) Record schemata which appear between string and each string in s_2 with matching bits no more than two bits apart (otherwise you have a separate schema).
   (d) Do not store duplicates.
   (e) Store schema frequency, sf, by running through all the strings in s_2 again.
3. Check to see what schemata are present randomly. For each string in sample s_1:
   (a) Use same random sample s_2 as in 2a).
   (b) Count the number of matches at each bit between the string and every string in s_2.
   (c) If
   $$sf < \prod_{i=1}^{\#definedbits} \frac{\#matches_i}{|s_2|}$$
   where sf is the schema frequency, $\#matches_i$ is the number of matches found in 2b) for bit i defined in schema,
   then remove the schema from the list.

For example, let $s_1 = \{0110100011\}$ and $s_2 = \{1110011101, 0101010101\}$. For step (2),

$s_{11} \cap s_{21} = \#110\#\#\#\#\#1$

with schemata

$\#110\#\#\#\#\#\#$ of frequency $\frac{1}{2}$ and

$\#\#\#\#\#\#\#\#1$ of frequency $\frac{2}{2}$

and

$s_{11} \cap s_{22} = 01\#\#\#\#0\#\#1$

with schemata

$01\#\#\#\#\#\#\#\#$ of frequency $\frac{1}{2}$ and

$\#\#\#\#\#\#0\#\#1$ of frequency $\frac{1}{2}$.

For step (3), the number of matches at each bit is

(1)(2)(1)(1)(0)(0)(1)(0)(0)(2).

Then for schema

$\#110\#\#\#\#\#\#$, $\frac{1}{2} < \frac{2 \cdot 1 \cdot 1}{2}$ so remove this schema,

$\#\#\#\#\#\#\#\#1$, $\frac{2}{2} \geq \frac{2}{2}$ so keep this schema,

$01\#\#\#\#\#\#\#\#$, $\frac{1}{2} < \frac{1 \cdot 2}{2}$ so remove this schema,

and

$\#\#\#\#\#\#0\#\#1$, $\frac{1}{2} < \frac{1 \cdot 2}{2}$ so remove this schema.

For the schemata in the pattern population, both samples are the entire pattern population. That is, all useful pattern schemata that can be located by the algorithm are found and counted. For the recognizer population, the $(\frac{1}{3} \cdot recognizerpopulationsize) = n$ most fit strings are chosen for the first sample, s_1, and the same size sample is chosen for the second random sample, s_2. So recognizer schemata occuring in $\frac{1}{n}$ of the recognizer population are located this way in every time step. Rather than being explicitly represented anywhere in the model, the schemata so-located are printed out at every time step so that they can be analyzed.

This model for pattern recognition is very similar to, though implemented differently from, Booker's genetic algorithm simulation[1]. In that simulation, the classifier system learns useful categories or taxa (the conditions of the classifers) for one or more categories in the environment, where the categories are generating input messages to the system. The taxa are equivalent to schemata though there are fewer of them in a classifier string than in the recognizer binary bit string. The binary bit string messages are equivalent to binary bit pattern strings that the classifers and recognizers, respectively, are trying to match.

The matchscore that he uses, called M_3, is defined for a taxon x and message (binary bit string) x' as

$$M_3(x) = \begin{cases} 1 & \text{if } x \text{ and } x' \text{ match identically} \\ \frac{l-n}{l^2} & \text{otherwise} \end{cases}$$

where n is the number of mismatched bit values and l is the length of the strings. This function is linear with slope $-\frac{1}{l^2}$ except for the case of all defining bits matching where M_3 returns the value 1 rather than $\frac{1}{l}$. This jump when the optimal solution is reached ensures that a good taxon is never lost from the population. On the other hand, the fitness function in the model just presented is an exponential curve which increases the fitness exponentially with each additional matching bit rather than just a big jump when the perfect match is found. The fitness function encourages bit by bit additions to matching schemata whereas the matchscore encourages this but really emphasizes the final perfect match. The fitness function is also more apt to keep schemata of all lengths in the population in case the pattern strings in the environment change so that the recognizer population can quickly build a matching recognizer from a few schemata. Matchscore just tries to match the categories in the current environment.

In the pattern recognition model, there are no mating restrictions, unlike Booker's simulation where two strings can mate only if they match the same message. Mating in the new model is probabilistic with respect to fitness, which is computed with respect to a sample of pattern strings. Thus, higher fitness implies that the recognizer string matches more pattern strings out of its sample which implies that there is a good chance that two mating strings both match at least one same pattern string, like a probabilistic mating restriction. This chance is equal to $1 - \frac{(a-m)(a-m-1)\cdots(a-2m+1)}{(a)(a-1)\cdots(a-m+1)}$, where a is the pattern population size and m is *matchsample*. For instance, in this model the chance would be $\frac{17}{24}$ where $a = 10$ and $m = 3$. Thus, the recognizer population can find the near-optimal points representing the pattern strings and still incorporate new strings with new schemata for each time step.

Crowding is also implemented similarly to Booker's simulation. The $\frac{1}{fitness}$, equivalent to $\frac{1}{strength}$, is used to pick out a *sample* of strings to delete based on which string best matches the new string. This is similar to Booker's crowding scheme where taxation and a tax rebate lower the strength of strings in a subpopulation that has reached its "carrying capacity". Then the strings with lower strength, are more likely to get removed, just as strings in the new model with lower $fitness$ and higher $\frac{1}{fitness}$ are more likely to get removed. Since fitness is noisy, there are recognizer strings in each subpopulation with lower fitness that will probably get removed; if the subpopulation is of size $\frac{1}{n}$, then these strings with lower fitness will appear in the sample and get chosen to be replaced. Also, in the current model, using a sample size equal to the number of desired subpopulations is simpler to implement than Booker's scheme where the system keeps track of which taxa are relevant to what message, unless the relevance criterion is such that it can be calculated for each subpopulation rather than every individual taxon.

Thus, the recognizer population in the pattern recognition model, just like the classifiers in Booker's model, can also find the near-optimal points representing the pattern strings and still incorporate new recognizer strings with new schemata for each time step.

4 Results

Preliminary runs have found parameter settings with which the recognizer population increases its fitness (equal to the sum of the fitnesses of each recognizer string) as shown in the following $log_{10}(fitness)$ vs. *time* plot. Increasing fitness implies that the recognizer population is matching one or more pattern strings at more bits.

To test the model's ability to solve the recognition problem, fifty recognizer strings have been run under the genetic algorithm to get them to match one pattern string. The population fitness does increase exponentially (Figure 1) until it reaches the maximum population fitness possible when about ninety percent of the recognizer strings match the pattern string at every bit position. The other ten percent match at almost every bit.

To test the model's ability to solve the covering problem, a run where fifty recognizer strings try to recognize ten pattern strings has been tried. The fitness of the recognizer population does increase fairly monotonically (Figure 2). The listing of pattern schemata found in the final recognizer population shows that all pattern schemata with five or fewer defining bits have also been found in the final recognizer population. Also, upon closer inspection of the recognizer population, a recognizer string can be found for each pattern string such that the recognizer string matches the pattern string at no fewer than five bit positions. Thus, the recognizer population recognizes and covers all of the pattern strings.

With what sample sizes does this genetic algorithm solve the recogntion and covering problems? How does the recognizer population evolve with these parameter settings?

It turns out to be critical that the value for *matchsample* be less than the size of the pattern population. That is, fitness for each string must be computed with respect to only a few patterns and with respect to different patterns each time step for a given recognizer string. If all patterns are used to determine the fitness of a recognizer string, then the entire recognizer population ends up matching only one pattern as it reaches maximum possible value for fitness.

There are several reasons why *matchsample* should be just a portion of the pattern population size. First of all, the computations to determine fitness are done faster if only a few patterns are used for the calculation.

Secondly, each recognizer uses the same sample of patterns to compute its fitness, namely all of the patterns. Thus, the first pattern with a longer schema found in the recognizer population will be the one matched by all of the recognizers over time. That schema will increase the fitness exponentially of the recognizer containing it with respect to the fitnesses of the other recognizers. That recognizer's descendents will take over the

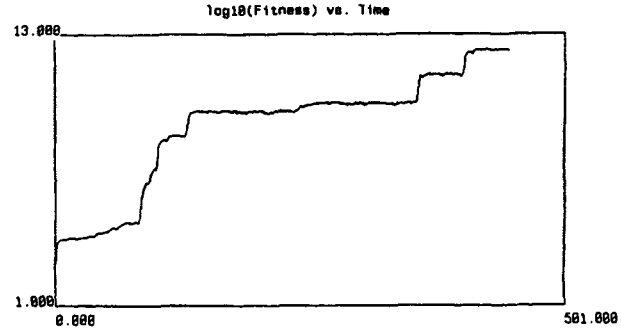


Figure 1. *patternpopulationsize* = *matchsample* = *crowdsample* = 1.

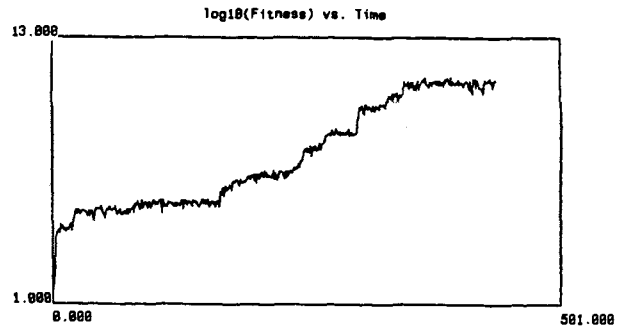


Figure 2. *patternpopulationsize* = *crowdsample* = 10, *matchsample* = 3.

entire recognizer population to match that one pattern. Thus, in order to cover all of the patterns, a different sample of patterns must be used to calculate fitness for each recognizer string.

Thirdly, and most importantly, sampling is the counterpart in this model of collision dynamics. One recognizer can try to recognize only a few patterns at any one time in real physical space. Modelling this accurately models the fact that this system is far from equilibrium most of the time where changes in the system are most likely to increase rather than decrease order in the system. That is, new schemata introduced into the recognizer population will get their numbers increased with respect to their fitness and give rise to new recognizer strings as they get distributed throughout the recognizer population. Otherwise, with non-noisy fitness, these new schemata would be ignored and eventually be removed from the recognizer population as one recognizer string containing one longer schemata was the only one with surviving descendent strings.

The other sample size for which the value turns out to be critical is *crowdsample*. With *crowdsample* = 10 (equal to *patternpopulationsize* in this case), a schema will be replaced if it is in $\frac{1}{10}$ of the recognizer population. This allows more room for other schemata to occur and distribute in $\frac{1}{10}$ of the recognizer population. If *crowdsample* is less than that, for example if it set to 3, then the fitness of the recognizer population increases

but only recognizes about three of the patterns.

With *crowdsample* = 10, as in the second run described, it was found that the recognizer population maintains a high number of schemata at all times, keeping the number of recognizer schemata around 100 ± 25 each time step. By maintaining such diversity in the recognizer population, all pattern schemata of 5 or fewer defining bits are found by the genetic algorithm along with a few schemata with even more defining bits. This is also due to the *recognizerpopulationsize* parameter. Theory predicts that when there are 2^n schemata of length n, a population of size $2^5 = 32$ or greater should be able to search these. A population of size 50, where $2^5 < 50 < 2^6$, does this plus finds a few other longer schemata. One of these longer schemata, not in the initial recognizer population, has been created by putting together two shorter schemata that increase in frequency enough to form the longer schema, which then increases in frequency. The shorter schemata decrease in frequency since they have been incoporated into the longer schema. Thus, we see the beginning of a default hierarchy forming[7][9]. First shorter, more general schemata are tested against the environment, and then the useful ones are used to build longer, more specific schemata that predict the environment (or in this case cover the strings in the pattern population even more accurately). With a larger population, more of the longer schemata should emerge.

5 Conclusion

The recognition and covering constraints are the minimal ones that the current model can satisfy. It does this by taking two shorter schemata and applying crossover to the strings of those schemata to form a new longer schema. This longer schema is a combination of the two shorter ones and is in a string that matches a pattern at more bits. Thus, a default hierarchy is emerging in the recognizer population. Mutation also can create such longer schemata and fitter strings but only one more bit will match an pattern. The one additional matching bit does not increase the fitness of the string nor the number of recognizer strings to match pattern strings as much as recombination does. However, these results are based on preliminary runs and need to be quantified more precisely before more specific conclusions can be made.

There are still many questions to study about this pattern recognition model.

1. How is the default hierarchy of schemata represented in the recognizer string population? Are there recognizer strings that contain several shorter pattern schemata and some recognizer strings with longer schemata? In terms of knowledge representation[6], is the representation fine-grained with longer, more specific schemata or coarse-grained with more general schemata? Is the entire emerging default hierarchy maintained in the recognizer population?

2. Do distinct subpopulations evolve in the recognizer population with respect to the patterns or other criteria such as regularities in the pattern strings? Does each recognizer string contain schemata for one pattern or for more than one pattern? If the *patternpopulationsize* is increased, will the system no longer be able to pick out all of the regularities which may no longer come from one pattern but may include several patterns? How will this depend on the *recognizerpopulationsize*?

3. How do the schemata in the recognizer population change when the initial pattern population is changed? And if one of the parameter values is changed? What if the fitness function is changed to give maximum fitness value when T bits match?

4. Are *matchsample* and *crowdsample*, which maintain diversity in the recognizer population, also resulting in suboptimal fitnesses for the recognizers that cover the patterns? That is, does it become impossible for perfect matches to evolve for each pattern? Is this somehow an adaptive advantage even if it is not a performance advantage? How can their values be set by some feedback mechanism based on regularities found and the resulting fitness, so that the recognizer population can continue to recognize and cover a pattern population that is changing its size and strings over time?

5. Is the final recognizer population robust to changes in the pattern population? Will the recognizer population fitness decrease alot quickly if an original pattern is replaced with a new and different pattern string? Will the entire recognizer population have to re-evolve to redistribute the pattern schemata it has, lose the ones no longer in the pattern population, and acquire new ones?

Various tools still need to be built to analyze the evolution of schemata and the default hierarchy in the recognizer population. For instance, since the schema finding algorithm only finds a sample of the schemata present in the pattern population, a reasonably-sized phylogenetic tree can be plotted over time based on what pattern schemata in the recognizer strings give rise to which new and possibly longer pattern schemata in new recognizer strings. The frequency of each pattern schemata in the recognizer population (equal to the number of recognizer strings with that pattern schemata) could be plotted over time and used with the phylogenetic tree to trace the development of the default hierarchy.

The frequencies of the schemata can also be studied with analytical tools developed for understanding dynamical systems. Every schemata changes its frequency (equal to the number of strings in the population belonging to that schema) in the population with respect to the probability

$$P_s(t+1) = \frac{M_s(t+1)}{M} = \hat{f}_s(t) \cdot \varepsilon_s \cdot M_s(t)$$

where schema s has an average expected fitness $\hat{f}_s(t)$ at time t, $M_s(t)$ is the number of strings in the recognizer population that are elements of schema s, M is the total number of strings in the fixed sized recognizer population, and ε_s is an error term created by having a schema instance get broken up by a crossover point or mutation. The size of the recognizer population and the sample sizes also affect the number of schemata and the number of their instances that are possible to have in the recognizer population. Thus, the schemata dynamics can be studied under different parameter settings for the population and sample sizes, and the affects of fluctuations occuring when new schemata are introduced into the recognizer population by crossover, mutation, or changes in the pattern population can also be studied.

Once the structure and evolution of the default hierarchy is better understood, this pattern recogniton model can be used to study the structural evolution of memory in other physical and biological systems that do pattern recognition. The features of the particular system can be assigned to the bits of the pattern and recognizer strings. Then one can study how the recognizer strings of that system's memory are structured with respect to the regularities in the patterns the system is trying to recognize.

For instance, in modeling the immune system, experimental evidence can be used to set up the fitness function and parameter values to be more realistic with real immune systems and to see if the model results in antibodies with similar characteristics as those found in the experiments. Some experimental studies have shown that one antibody can recognize more than one antigen (called "multi-specificity" in immunology) [13]. Such an antibody is a generalist. Other evidence shows that when new antigens are introduced, new antibodies are created by an increased rate of point mutations. The affinity of these new antibodies for new antigens increases ten-fold over a short period of time, indicating that the antibodies become very specific at recognizing the antigens [14]. Thus, some sort of default hierarchy is developing in the immune system. What is the nature of this default hierarchy if the fitness function and paramters are set to correspond to actual numbers used and seen in these experiments? What antibodies result if the intial antibodies are ones that have evolved in this pattern recognition model and can only undergo mutation in the future when presented with a new antigen? How might a network of such antibodies compensate for any shortcomings of the antibody population which can no longer undergo crossover?

Also, in a letter recognition system, letter patterns can be represented with line features of different lengths, angles, and positions. Could this model then generate a default hierarchy to recognize the letter or letters? How could it recognize the same letters if they are written in a different style?

In cognitive science, a concept is represented by a prototype developed from a set of instances. The nature of the prototype with respect to the properties of the instances is not yet clearly understood, but the instances do not necessarily all share some set of property values which might then be the prototype. Rather, each instance shares one or more property values with one or more other instances in the set [11]. Translated into the pattern recognition model, the recognizer population can try to match the instances in the environment where the prototype would appear as the more general schema that persists over time and is not just used to build one specific schema. On the other hand, the prototype might turn out to be some piece of the default hierarchy distributed over several recognizer strings. How would this correspond to any experimental data that has been collected by psychologists on the nature of prototypes? What kinds of prototypes are actually communicated in these experiments (i.e. trasmitted verbally to the experimenter)?

By understanding the organization of the default hierarchy in the recognizer strings, the memories of other pattern recognition systems evolving under a genetic algorithm can be analyzed as well. A system that tries to recognize patterns in the environment and then builds an internal model or memory of those patterns must use some definition of similarity to cluster the patterns into classes. If this system also transmits its internal model by a code representing how to reconstruct that model, then this system is a good candidate to be cast in and analyzed by this pattern recognition model.

6 Acknowledgements

This work has been performed under the auspices of the U.S. Department of Energy. I would like to thank John H. Holland for his guidance and ideas about genetic algorithms and modelling, Chris Langton for his comments on modelling artificial life, and Alan Perelson, Doyne Farmer, and Norman Packard for their valuable insights into the structure and dynamics of biological and physical systems.

References

[1] Booker, L.B. 1985, "Improving the Performance of Genetic Algorithms in Classifier Systems". Proceedings of an International Conference on Genetic Algorithms and Their Applications, Pittsburgh, Pennsylvania : Carnegie-Mellon University.

[2] DeJong, K.A. 1975, "Analysis of the Behavior of a Class of Genetic Adaptive Systems". PhD Dissertation, Ann Arbor : The University of Michigan.

[3] Farmer, J.D., Kauffman, S.A., and Packard, N.H. 1987, "Autocatalytic Replication of Polymers". Physica 22D, pp. 50-67.

[4] Farmer, J.D., Kauffman, S.A., Packard, N.H., and Perelson, A.S. 1987, "Adaptive Dynamic Networks as Models for the Immune System and Autocatalytic Sets". Annals New York Academy of Sciences.

[5] Farmer, J.D., Packard, N.H., and Perelson, A.S. 1986, "The Immune System, Adaptation, and Machine Learning". Physica 22D, pp. 187-204.

[6] Feldman, J.A. 1986, "Neural Representation of Conceptual Knowledge". TR189, Dept. of Computer Science, Rochester, NY : The University of Rochester.

[7] Goldberg, D.E. 1983, "Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning". PhD Dissertation, Ann Arbor : The University of Michigan.

[8] Holland, J.H. 1975, Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan : The University of Michigan Press.

[9] Holland, J.H. 1985, "Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule Base Systems". Machine Learning II, Ch. 20, Los Altos, CA : Morgan Kauffman.

[10] Jerne, N.K. 1974, "Toward a Network Theory of the Immune System". Annals of Immunology (Inst. Pasteur) 125C : 373.

[11] Kaplan, S. 1986, Neural Models, Course Notes, Ann Arbor : The University of Michigan.

[12] Langton, C.G. 1984, "Self-Reproduction in Cellular Automata". Physica 10D, pp. 135-144.

[13] Rosenstein, R.W., Musson, R.A., Armstrong, M.Y.K., Konigsberg, W.H., and Richards, F.F. 1972, "Contact Regions for Dinitrophenyl and Menadione Haptens in an Immunoglobulin Binding More Than One Antigen". Proceedings of the National Academy of Science, USA, Vol. 69, No. 4, pp. 887-881.

[14] Wysocki, L.J., Manser, T., and Gefter, M.L. 1986, "Somatic Evolution of Variable Region Structures During an Immune Response". Proceedings of the National Academy of Science, USA, Vol. 83, pp. 1847-1851.

An Adaptive Crossover Distribution Mechanism for Genetic Algorithms

J. David Schaffer
Amy Morishima

Philips Laboratories
North American Philips Corporation
Briarcliff Manor, New York

ABSTRACT

This paper presents a new version of a class of search procedures usually called genetic algorithms. Our new version implements a modified string representation that includes special punctuation used by the crossover recombination operator. The idea behind this scheme was abstracted from the mechanics of natural genetics and seems to yield a search procedure wherein the action of the recombination operator can be made to adapt to the search space in parallel with the adaptation of the string contents. In addition, this adaptation happens "for free" in that no additional operations beyond those of the traditional genetic algorithm are employed.

We present some empirical evidence that suggests this procedure may be as good as or better than the traditional genetic algorithm across a range of search problems and that its action does successfully adapt the search mechanics to the problem space.

1. Background

A genetic algorithm is an exploratory procedure that is able to locate high performance structures in complex task domains. To do this, it maintains a set (called a population) of trial structures, represented as strings. New test structures are produced by repeating a two-step cycle (called a generation) which includes a survival-of-the-fittest selection step and a recombination step. Recombination involves producing new strings (the offspring) by operations upon one or more previous strings (the parents). The principle recombination operator was abstracted from knowledge of natural genetics and is called crossover. Holland has provided a theoretical explanation for the high performance of such algorithms [5] and this performance has been demonstrated on a number of complex problem domains such as function optimization [3, 4] and machine learning [6, 8, 9].

The action of the traditional crossover is illustrated in figure 1. Starting with two strings from the population†, a point is selected between 1 and L-1, where L is the string length. Both strings are severed at this point and the segments to the right of this point are switched. The two starting strings are usually called the parents and the two resulting strings , the offspring. Taking the metaphor one step further, we will call this operation a mating with a single crossover event. In all previous work with which we are familiar, the crossover point is chosen with a uniform probability distribution.

The motivation for our new crossover operator sprang from some properties of this traditional operator. Specifically, it has a known bias against properly sampling structures which contain coadaptive substrings that are far apart. In metaphorical terms, we might call them genes which are far apart on the chromosome. The reason is not difficult to grasp intuitively. The farther apart the genes are, the higher the probability that a uniformly selected random crossover point will fall between them causing them to be passed to different offspring. We observe that this crossover operator requires only knowledge of the string length; it pays no attention to its contents. Furthermore its action is nonadaptive. It performs the same way in every generation.

In contrast to this, what we know of Nature's genetic crossover activity suggests that the location of crossover events may be quite sensitive to the contents of the chromosome [1]. There are many activities in this microworld which involve the initiation of an action by the binding of an enzyme to a specific base-sequence. We were motivated to design a crossover mechanism which would adapt the distribution of its crossover points by the same survival-of-the-fittest and recombination processes already in place. We reasoned that it should do so by the use of special punctuation marks introduced into the string representation for this purpose. The operation envisioned for this new crossover was to proceed by "marking" the site of each crossover event in the string in which it occurred. Thus if the search space had the characteristic that crossovers at particular loci were consistently associated with inferior offspring, then they would die out, taking their markings with them. The converse was also hoped for. If no consistent relation existed to be exploited, then a

† We will show all strings as bit strings, but this is not a requirement imposed by the algorithm.

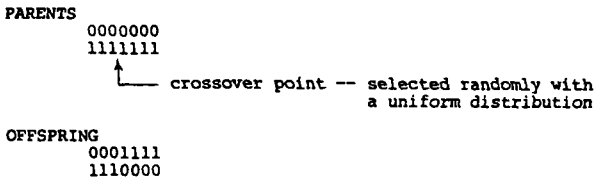


Figure 1. The action of the traditional crossover operator. The two parent strings are shown as all zeros and ones for clarity.

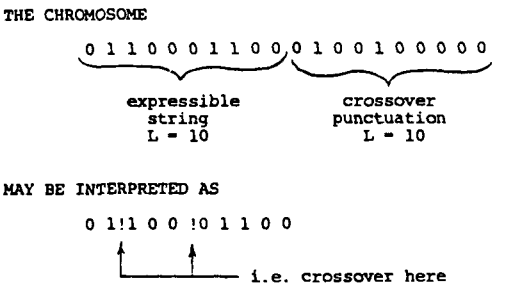


Figure 2. The string representation of chromosomes with crossover punctuation marks.

random allocation of markings was expected. In this, we were reminded of speculation by Lenat that modern gene pools may contain accumulated heuristics to guide genetic search as well as an accumulation of well-adapted expressible genes [7].

The idea of punctuation in the strings used by a genetic algorithm was first proposed by Holland [5], but this proposal was for a different purpose than that proposed here.

The rest of this paper will present, the mechanics of our new crossover operator, some empirical results indicating that superior performance can indeed be achieved with it, and some data exhibiting properties of its behavior.

2. Mechanics

In this section we will explain the mechanics of operation of our new crossover operator. We address how the crossover punctuation is coded, how an initial distribution is generated, how this distribution affects the crossing over of the functional parts of the strings, how the punctuation marks themselves are passed on to the offspring, and how a linkage is established between individual punctuation marks and the functional substrings with which they are associated.

The representation is straightforward. To the end of each chromosome (string of bits interpretable as a point in the search space) we attach another bit string of the same length. Thus any string representation previously used with a traditional genetic algorithm can be employed in our scheme simply by doubling its length. The bits in the new section are interpreted as crossover punctuation, 1 for yes and 0 for no (i.e. 1=crossover, 0=no crossover). The loci in the punctuation (second) part of the string correspond one-to-one with the loci in the functional (first) part of the chromosome. Thus it is natural to think of these two parts of the chromosome as interleaved. Punctuation mark i tells whether crossover is or is not to occur at locus i. In figure 2, the punctuation marks are shown as "!" in the functional string.

It is common practice when beginning a genetic search to initiate a population of strings by randomly generating bits with equal probability for zero and one. We follow this practice for the functional string, but the probability of generating a one in the punctuation string is designated P_{2o} and is set externally. The influence of this variable was studied empirically.

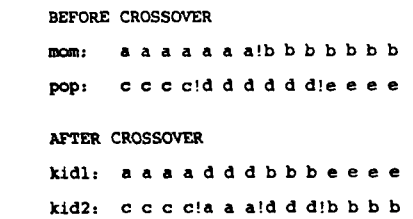


Figure 3. The action of punctuated crossover.

The mechanics of crossover governed by these punctuation marks is illustrated in figure 3. The bits from each parent string are copied one-by-one to one of the offspring from left to right. When a punctuation mark is encountered in either parent, the bits begin going to the other offspring (a crossover). When this happens, the punctuation marks themselves are also passed on to the offspring, just before the crossover takes effect. Thus we may think of the marks as being linked to the functional bit string to the left of its locus. A little experimentation with pencil and paper should serve to give the reader a sense of the redistribution possibilities of this process. Some parental distributions will result in all the punctuation marks being passed to one of the offspring and none to its sibling, while others will redistribute clumped distributions. When an offspring fails to survive the fitness-based selection step in its generation, its punctuation marks die with it. Thus, the dynamics of the distribution of these marks in the gene pool should reflect an accumulating experience about where is or is not a good place to crossover the genetic material in the pool.

The action of the mutation operator has traditionally been employed as a low level (i.e. small probability) defense against premature convergence of the gene pool. It seems consistent with our metaphor to allow the mutation operator equal access to the entire chromosome, both functional and punctuation parts. A few experiments supported this belief.

3. Empirical Evidence

We selected the task domain of function optimization (minimization) to test the capabilities of this new genetic algorithm and a set of five scalar-valued functions which has been used in the past to test genetic algorithms. These functions provide a range of characteristics of search problems and are summarized in table 1. Recently Grefenstette has found a configuration of the traditional genetic algorithm which performs consistently better on this function set than any previously known. [4]. We will use his results (which we will call BTGA for best traditional genetic algorithm) as a benchmark.

TABLE 1
Functions Comprising the Test Environment

Fcn	Dimensions	Space Size	Description
f1	3	1.0×10^{9}	parabola
f2	2	1.7×10^{6}	Rosenbrock's saddle
f3	5	1.0×10^{15}	step function
f4	30	1.0×10^{72}	quadratic with noise
f5	2	1.6×10^{10}	Shekel's foxholes

We adopted Grefenstette's strategy of setting a genetic algorithm to optimize a genetic algorithm (GA) in order to locate a good set of parameter values for our new procedure (a "meta-search"). The meta-level GA was a traditional GA that allowed vector-valued fitness (one dimension for each of the five functions) called VEGA [8]. The parameter set searched at this level included: population size, crossover rate, mutation rate, P_{so}, and scaling window. The performance measure was online average function value (i.e. an average of all trials in a run of 5000 function evaluations). For more details on these matters the reader is referred to Grefenstette's paper and its predecessors.

Since the performance of BTGA on each of the individual functions was not previously published, we first estimated this by running a BTGA on each one five times (n) with different random seeds. The results are given in table 2.

The results of the Meta-search revealed that the genetic algorithm with punctuated crossover (GAPC) performed well for the whole set of functions at the following parameter settings: population size = 40, crossover rate = 0.45, mutation rate = 0.002, P_{so} = 0.04, and scaling window = 3. Estimates of the performance of GAPC comparable to those for BTGA are given in table 3 along with results of two-tailed t-tests of the mean differences between them.

TABLE 2
Performance of Best Traditional Genetic Algorithm on Test Functions

Function	mean	s.d.	n	global optimum
f1	1.664	.1900	5	0
f2	25.16	4.497	5	0
f3	-27.78	.2111	5	-30
f4	24.28	1.383	5	0
f5	30.78	2.148	5	≈1

TABLE 3
Performance of Genetic Algorithm with Punctuated Crossover on Test Functions and a comparison with the Best Traditional Genetic Algorithm

Function	mean	s.d.	n	t-test	significance
f1	1.111	.1706	5	4.84	.01
f2	17.22	3.279	5	3.19	.05
f3	-27.90	.5907	5	0.42	ns
f4	20.32	1.320	5	4.63	.01
f5	14.81	1.475	5	13.71	.001

These results clearly show the superiority of GAPC. It statistically outperforms BTGA on four of the five functions and, is no worse on the other (f3). We believe the reason for this latter result lies in a floor effect. Both GAs find good solutions very quickly on f3 so that the online average rapidly approaches the global optimum. There is simply insufficient room for improvement to allow for a statistical difference. We believe the same explanation applies to the finding that no significant differences were noted between BTGA and GAPC when offline average was used as a criterion.

4. Characterization of GAPC Search

There are a number of questions about the realization of the performance characteristics we envisioned when this scheme was designed. Specifically, does the distribution of crossover marks adapt to the task environment in a meaningful way, is the process stable (i.e. does the number of punctuation marks in the population tend to vanish or saturate), and how many crossovers per mating does it settle upon (if it does settle)? In this section we present some evidence related to these questions that was collected while monitoring the searches reported above.

We define the population distribution of punctuation marks at time t as the sum of the punctuation bits at each locus across all the individuals in the population.

$$p_{so}(l,t) = \sum_{i=1}^{popsize} punct_i(l,t) \qquad (2)$$

The total number of punctuation marks in the population is then

$$T_{so}(t) = \sum_{l=1}^{L} p_{so}(l,t) \qquad (3)$$

Figure 4 shows a time history of $p_{so}(l,t)$ for 200 generations of one search of function f1. The x-axis is chromosome locus and the y-axis is both $p_{so}(l,t)$ and time

(generations). The vertical separation between successive generations is equal to popsize (i.e. the number of individuals in the population) so that, a locus at which every member of the population has a punctuation mark, will appear as a peak which just reaches the baseline of the line above. This figure shows that the initial distribution is flat since the random initialization of the punctuation marks does not favor one locus over any other. As time progresses, however, some loci tend to accumulate more punctuation marks than others. The location of these concentrations changes with time; a peak may appear and remain prominent for some generations only to die out as others emerge. The distribution of punctuation marks does indeed seem to adapt as the gene pool adapts.

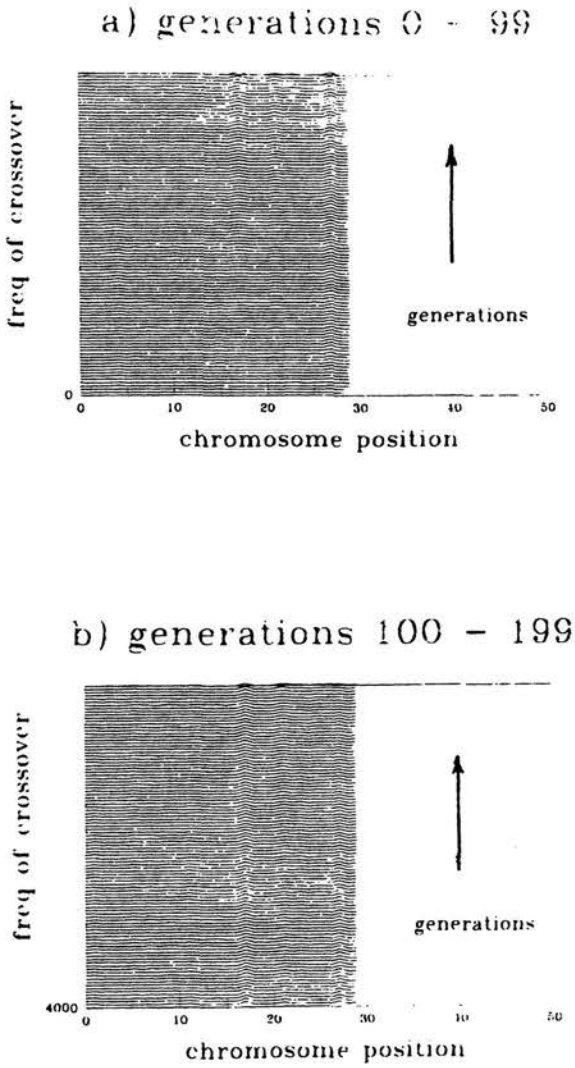


Figure 4. A time history of the distribution of punctuation marks for one run on f1.

Figure 5 shows a plot of $T_{20}(t)$ for the same run. Also shown are the $\max p_{20}(l,t)$ and $\min p_{20}(l,t)$. Although the total seems to be growing with no sign of stabilizing, the population is far from saturated†, the minimum remains zero until the last few generations and the maximum concentration at any one locus never exceeds 30 out of the possible 40 (i.e. popsize).

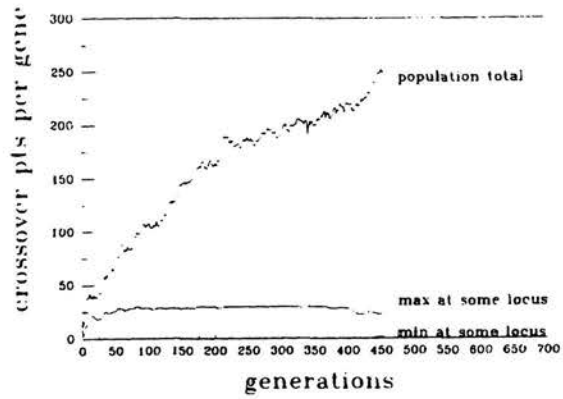


Figure 5. How the numbers of punctuation marks change with time. These data were from one search on f1, and were typical.

Figure 6 shows the average number of crossover events occurring per mating. This simple counting can be misleading, however, since the gene pool is converging as the generations progress. Note that when a crossover event swaps gene segments which are identical, it is an unproductive crossover event. When these events are discounted, we were surprised to see that the number of "productive" crossover events per mating remained nearly constant. What is more, the level at which this statistic holds appears to correlate strongly with L. See table 4. These results are not dissimilar to results reported by DeJong when experimenting with multiple crossover points [2]

TABLE 4
Time-averaged "Productive" Crossovers per Mating

Function	chromosome length	crossovers
f1	30	1.49
f2	24	0.87
f3	50	2.02
f4	240	8.64
f5	32	1.65

† Saturation would mean a punctuation bit at every one of the *popsize* $\times L$ ($40 \times 30 = 1200$) possible locations.

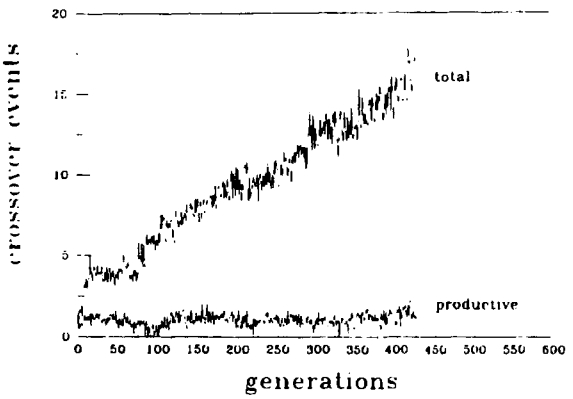


Figure 6. Total and "productive" crossover events per mating for one run on f1.

5. Conclusions

We have described a modified knowledge representation and crossover operator for use with genetic search. Its design was driven by intuition abstracted from Nature's mechanisms of crossover during meiosis. Experiments indicate that it performs as well or better than a traditional GA for a set of test problems, that exhibits a range of search space properties. Experiments on other test problems are continuing.

The distribution of crossover events evolves as the search progresses and the statistics of "productive" crossover events per mating indicate steady search effort even in the face of a converging gene pool. These statistics seem to correlate with chromosome length and are consistent with previous results.

We remain cautiously optimistic that continued experimentation will strengthen these conclusions and will lead to a robust approach to adaptive knowledge representation.

Acknowledgement

We wish to acknowledge the valuable contributions of D. Paul Benjamin to the conception of this project and to discussions of its implications.

References

1. B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts and J. D. Watson, *Molecular Biology of the Cell*, Garland Publishing, Inc., New York, 1983.
2. K. A. De Jong, Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, 1975.
3. K. A. De Jong, Adaptive System Design: A Genetic Approach, *IEEE Transactions on Systems, Man & Cybernetics SMC-10*,9 (September 1980), 566-574.
4. J. J. Grefenstette, Optimization of Control Parameters for Genetic Algorithms, *IEEE Transactions on Systems, Man & Cybernetics SMC-16*,1 (January-February 1986), 122-128.
5. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
6. J. H. Holland and J. S. Reitman, Cognitive Systems Based on Adaptive Algorithms, in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth (editor), Academic Press, New York, NY, 1978.
7. D. B. Lenat, The Role of Heuristics in Learning by Discovery: Three Case Studies, in *Machine Learning*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (editor), Tioga, Palo Alto, CA, 1983.
8. J. D. Schaffer, Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms, Ph.D. Thesis, Department of Electrical Engineering, Vanderbilt University, December 1984.
9. S. F. Smith, Flexible Learning of Problem Solving Heuristics Through Adaptive Search, *8th International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, August 1983.

GENETIC ALGORITHMS WITH SHARING FOR MULTIMODAL FUNCTION OPTIMIZATION

David E. Goldberg, The University of Alabama,
Tuscaloosa, AL 35487

and

Jon Richardson, The University of Tennessee
(formerly at The University of Alabama),
Knoxville, TN 37996

ABSTRACT

Many practical search and optimization problems require the investigation of multiple local optima. In this paper, the method of sharing functions is developed and investigated to permit the formation of stable subpopulations of different strings within a genetic algorithm (GA), thereby permitting the parallel investigation of many peaks. The theory and implementation of the method are investigated and two, one-dimensional test functions are considered. On a test function with five peaks of equal height, a GA without sharing loses strings at all but one peak; a GA with sharing maintains roughly equally sized subpopulations clustered about all five peaks. On a test function with five peaks of different sizes, a GA without sharing loses strings at all but the highest peak; a GA with sharing allocates decreasing numbers of strings to peaks of decreasing value as predicted by theory.

INTRODUCTION

Genetic algorithms (GAs) are finding increasing application in a variety of problems across a spectrum of disciplines (Goldberg & Thomas, 1986). This is so, because GAs place a minimum of requirements and restrictions on the user prior to engaging the search procedure. The user simply codes the problem as a finite length string, characterizes the objective (or objectives) as a black box, and turns the GA crank. The genetic algorithm then takes over, seeking near-optima primarily through the combined action of reproduction and crossover. These so-called simple GAs have proved useful in many problems despite their lack of sophisticated machinery and despite their total lack of knowledge of the problem they are solving. Yet as their usage has grown, several objections to their performance have arisen. Simple GAs have been criticized for sub-par performance on multimodal (multiply-peaked) functions. They have also been criticized for so-called premature convergence where substantial fixation occurs at most bit positions before obtaining sufficiently near-optimal points (Cavicchio, 1970; De Jong, 1975; Mauldin, 1984; Baker, 1985).

In this paper, we examine the first of these maladies and propose a cure borrowed from nature. In particular, our herbal remedy causes the formation of niche-like and species-like subdivision of the environment and population through the imposition of sharing functions. These sharing functions help mitigate unbridled head-to-head competition between widely disparate points in a search space. This reduction in competition between distant points thereby permits better performance on multimodal functions. As a side benefit we find that sharing helps maintain a more diverse population and more considered (and less premature) convergence. In the remainder of this paper, we review the problem and past efforts to solve it; we consider the theory of niche and speciation through Holland's modified two-armed bandit problem, and we compare the performance of a genetic algorithm both with and without the sharing function feature. Finally we examine extensions of the sharing function idea to permit its implementation in a wide array of problems.

MULTIMODAL OPTIMIZATION, GENETIC DRIFT, AND A SIMPLE GA

The difficulty posed by a multimodal problem for a simple genetic algorithm may be illustrated by a straightforward example. Figure 1 shows a bimodal function of a single variable: $f(x) = 4(x-0.5)^2$ coded by a normalized, five-bit binary integer. In this problem, the two optima are located at extreme ends of the one-dimensional space. If we start a genetic algorithm with a population chosen initially at random and let it run for a large number of generations, our fondest hope is that stable subpopulations cluster about the two optima (about 00000 and 11111). In fact if we perform this experiment, we find that the simple GA eventually clusters all of its points about one peak or the other.

Why does this happen? After all, doesn't the fundamental theorem of genetic algorithms (Holland, 1975; De Jong, 1975; Goldberg, 1986) tell us that exponentially increasing numbers of trials will be given to the observed best schemata? Yes it does, but the theorem assumes an infinitely large population size. In a finite size popula-

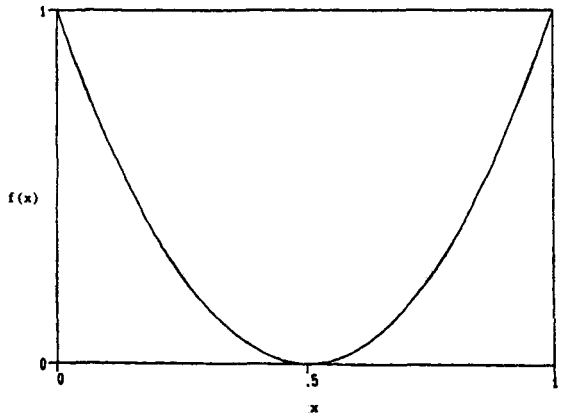


Figure 1. Bimodal function with equal peaks.

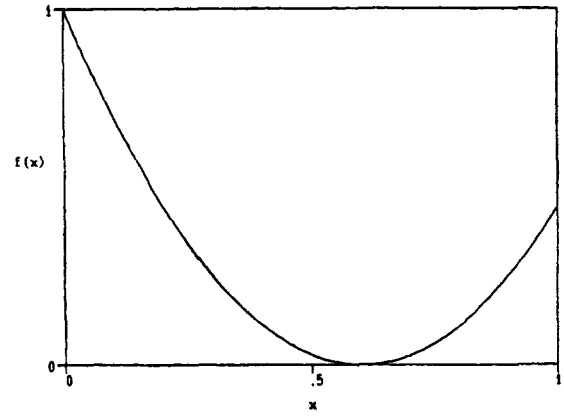


Figure 2. Bimodal function with unequal peaks.

tion, even when there is no selective advantage for either of two competing alternatives (as is the case for schemata 11*** and 00*** in the example problem) the population will converge to one alternative or the other in finite time (De Jong, 1975; Goldberg & Segrest, this volume). This problem of finite populations is so important that geneticists have given it a special name, genetic drift. Stochastic errors tend to accumulate, ultimately causing the population to converge to one alternative or another.

The convergence toward one optimum or another is clearly undesirable in the case of peaks of equal value. In multimodal problems where peaks of different altitudes exist, the desirability of convergence to the globally best peak is not so clear cut. In Figure 2 we see a bimodal function with unequal peaks: $f(x) = 2.8(x-0.6)^2$ with a five-bit normalized coding. If we are interested in obtaining only the global optimum, we should not mind the eventual convergence of the population to the leftmost point; however, this convergence is not always guaranteed. Small initial populations may allow sampling errors which overestimate the schemata of the rightmost points thereby permitting convergence to the wrong peak. Furthermore, in real world optimization we are often interested in having information about good, better, and best solutions. When this is so, it might be nice to see a form of convergence that permits stable subpopulations of points to cluster about both peaks according to peak fitness.

In either of these cases, we can argue for more controlled competition and less reckless convergence than is possible when we work with a simple, tripartite (reproduction, crossover, and mutation) genetic algorithm. For these reasons we turn to the theory of niche and speciation to find an appropriate model for naturally regulated competition.

THEORY OF SPECIES AND NICHE

The results of our initial gedanken-experimente (thought experiments) with simple GAs and multimodal functions are somewhat perplexing when juxtaposed with natural example. In our problem with equal peaks, the simple GA converges on one peak or the other even though both peaks are equally useful. By contrast, why doesn't nature converge to a single species? In our second problem with unequal peaks, we notice that the simple GA again converges to one peak, usually--but not always--the "correct" peak. How, when faced with a somewhat less fit species, does nature choose to limit population size before resorting to extinction? In both cases, nature has found a way to combat unbridled competition and permit the formation of stable subpopulations. In nature, different species don't go head to head. Instead they exploit separate niches (sets of environmental features) in which other organisms have little or no interest. In this section, we need to bridge the gap between natural example and genetic algorithm practice through the application of some useful theory.

Although there is a well-developed biological literature in both niche and speciation, its transfer to the arena of GA search has been limited. Like many other concepts and operators, the first theories directly applicable to artificial genetic search are due to Holland (1975). To illustrate niche and species, Holland introduces a modification of the two-armed bandit problem with distributed payoff and sharing. Let's examine his argument with a concrete formulation of the same problem.

Imagine a two-armed bandit as depicted in Figure 3. In the ordinary two-armed bandit problem (Holland, 1975; De Jong, 1975), we have two arms, a left arm and a right arm, and we have different payoffs associated with each arm. Suppose we have an expected payoff associated with the left arm of $25 and an expected payoff associated with the right arm of $75; in the standard two-armed bandit, we are unaware initially which arm pays the higher amount, and our dilemma is to min-

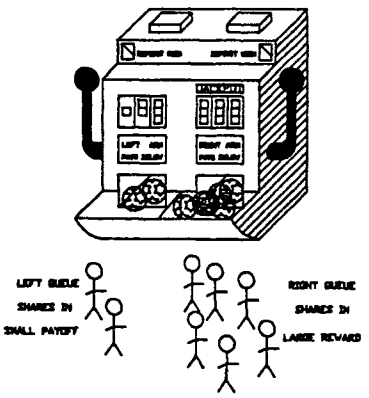


Figure 3. Sketch of the two-armed bandit with queues and sharing.

imize our expected losses over some number of trials. In this form, the two-armed bandit problem puts the tradeoffs between exploration and exploitation in sharp perspective. We can take extra time to experiment, but in so doing risk the possible gain from choosing the right arm, or we can experiment briefly and risk making an error once we choose the arm we think is best. In this way, the two-armed bandit has been used to justify the allocation strategy adopted by the reproductive plans of simple genetic algorithms.

This is not our purpose here. Instead we examine the modified two-armed bandit problem to put the concepts of niche and species in sharper focus. In the modified problem, we further suppose that we have a population of some number of players, say 100 players, and that each player may decide to play one arm or the other. If at this point we do nothing else, we simply create a parallel version of the original two-armed bandit problem where we expect that all players eventually line up behind the observed best (and actual best) arm. To produce the subdivision of species and niche, we introduce an important rule change. Instead of allowing a full measure of payoff for each individual, individuals who choose a particular arm are now forced to share the wealth derived from that arm with other players queued up at the arm. At first glance, this change appears to be quite minor. In fact, this single modification causes a strikingly and surprisingly different outcome in the modified two-armed bandit.

To see why and how the results change, we first recall that despite the different rules of the game, we still allocate population members according to payoff. In the modified game, an individual will receive a payoff which depends on the arm payoff value and the number of individuals queued up at that arm. In our concrete example, an individual lined up behind the right arm when all individuals are lined up behind that same arm receives an amount \$75/100 = \$0.75. On the other hand, an individual lined up behind the left arm when all individuals are queued there receives \$25/100 = \$0.25. In both cases, there is motivation for some individuals to shift lines. In the first case, a single individual changing lines stands to gain an amount \$25.00 - \$0.75 = \$24.25. The motivation to shift lines is even stronger in the second case. At some point in between we should expect there to be no further motivation to shift lines. This will occur when the individual payoffs are identical for both lines. If N is the population size, m_{right} and m_{left} are the number of individuals behind the right and left queues, and f_{right} and f_{left} are the expected payoff values from the right and left arms respectively, the equilibrium point may be calculated as follows:

$$\frac{f_{right}}{m_{right}} = \frac{f_{left}}{m_{left}}$$

In our example, this complete equalization of individual payoff occurs when 75 players select the right arm and 25 players select the left arm, because \$75/75 = \$25/25 = \$1.

This problem may be extended to the k-armed case directly, and the extension does not change the fundamental conclusions at all: the system attains equilibrium when the ratios of arm payoff to queue length are equal (Holland, 1975). The incorporation of forced sharing causes the formation of stable subpopulations (species) behind different arms (niches) in the problem. Furthermore, the number of individuals devoted to each niche is proportional to the expected niche payoff. This is exactly the type of solution we had hoped for when we considered the bimodal problems of Figures 1 and 2. Of course the extension of the sharing concept to real genetic algorithm search is more difficult than the idealized case implies. In a real genetic algorithm there are many, many arms and deciding who should share and how much should be shared becomes a non-trivial question. In the next section we will examine a number of current efforts to induce niche and species through indirect or direct sharing.

A BRIEF REVIEW OF CURRENT SCHEMES

A number of methods have been implemented to induce niche and species in genetic algorithms. In some of these techniques the sharing comes about indirectly. Although the two-armed bandit problem is a nice, simple abstract model of niche and species formation and maintenance, nature is not so direct in divvying up her bounty. In natural settings, sharing comes about through crowding and conflict. When a habitat becomes fairly full of a particular organism, individuals are forced to share available resources.

Cavicchio's (1971) dissertation study was one of the first to attempt to induce niche-like and species-like behavior in genetic algorithm search. Specifically, he introduced a mechanism he called

preselection. In this scheme, an offspring replaces the inferior parent if the offspring's fitness exceeds that of the inferior parent. In this way diversity is maintained in the population because strings tended to replace strings similar to themselves (one of their parents). Cavicchio claimed to maintain more diverse populations in a number of simulations with relatively small population sizes (n=20).

De Jong (1975) has generalized preselection in his crowding scheme. In De Jong crowding, individuals replace existing strings according to their similarity with other strings in an overlapping population. Specifically, an individual is compared to each string in a randomly drawn subpopulation of CF (crowding factor) members. The individual with the highest similarity (on the basis of bit-by-bit similarity count) is replaced by the new string. Early in the simulation, this amounts to random selection of replacements because all individuals are likely to be equally dissimilar. As the simulation progresses and more and more individuals in the population are similar to one another (one or more species have gotten a substantial foothold in the population) the replacements of individuals by similar individuals tends to maintain diversity within the population and reserve room for one or more species. De Jong has had success with the crowding scheme on multimodal functions when he used crowding factors CF=2 and CF=3. De Jong's crowding has subsequently been used in a machine learning application (Goldberg, 1983).

Booker (1982) discusses a direct application of the sharing idea in a machine learning application with genetics-based, classifier systems. In classifier systems, a sub-goal reward mechanism called a bucket brigade passes reward through a network of rules like money passing through an economy. Booker suggests that appropriately sized subpopulations of rules can form in such systems if related rules are forced to share payments. This idea is sound and has been forcefully demonstrated in Wilson's recent work with boolean function learning (Wilson, 1986); however, it does not transfer well to function optimization, because unlike classifier systems, there is no general way in function optimization to determine which strings are related.

Shaffer (1984) has used separate, fixed size subpopulations in his study of vector evaluated genetic algorithms (VEGA). In this study, each component of the vector (each criterion or objective measure) is mapped to its own subpopulation where separate reproduction processes are carried out. The method has worked well in a number of trial functions; however, Shaffer has expressed some concern over the procedure's ability to handle middling nondominated individuals--individuals that may be Pareto optimal but are not extremal (or even near extremal) along any single dimension. Furthermore, although the study does use separate subpopulations, it is unclear how the same method might be applied to the more usual single-criterion optimization problem.

A direct exploration of biological niche theory in the context of genetic algorithms is contained in Perry's (1985) dissertation. In this work, Perry defines a genotype-to-phenotype mapping, a multiple-resource environment, and a special entity called an external schema. External schemata are special similarity templates defined by the simulation designer to characterize species membership. Unfortunately, the required intervention of an outside agent limits the practical use of this technique in artificial genetic search. Nonetheless, the reader interested in the connections between biological niche theory and GAs may be interested in this work.

Grosso (1985) also maintains a biological orientation in his study of explicit subpopulation formation and migration operators. Multiplicative, heterotic (problems with diploid structures where a heterozygote is more highly fit than the homozygote) objective functions are used in this study, and as such, the results are not directly applicable to most artificial genetic search; however, Grosso was able to show the advantage of intermediate migration rate values over either isolated subpopulations (no migration) and panmictic (completely mixed) subpopulations. This study suggests that the imposition of a geography within artificial genetic search may be another useful way of assisting the forming diverse subpopulations. Further studies are needed to determine how to do this in more general artificial genetic search applications.

Although he has not directly addressed niche and species, Mauldin (1984) has attempted to better maintain diversity in genetic algorithms through his uniqueness operator. The uniqueness operator arbitrarily returns diversity to a population whenever it is judged to be lacking. To implement uniqueness, Mauldin defines a uniqueness parameter k_u that may decrease with time (similar to the cooling of simulated annealing). He then requires that for insertion in a population, an offspring must be different than every population member at a minimum of k_u loci. If the offspring is not sufficiently different, it is mutated until it is. By itself, uniqueness is little more than a somewhat knowledgeable (albeit expensive) mutation operator. That it is useful in improving offline (convergence) performance is not unexpected. Grefenstette (1986) has recently supported the notion of fairly high mutation probabilities (p_m = 0.01 to 0.1) when convergence to the best is the main goal. It is interesting to note that uniqueness combined with De Jong's crowding scheme worked better than either operator by itself (Mauldin, 1984). This result suggests that maintaining diversity for its own sake is not the issue. Instead, we need to maintain appropriate diversity--diversity that in some way helps cause (or has helped cause) good strings. In the next section, we show how we can maintain appropriate diversity through the use of sharing functions.

SHARING FUNCTIONS

In attempting to induce species, we must either directly or indirectly cause intraspecies sharing, but we are faced with two important questions: who should share, and how much should be shared? In natural systems, these two questions are answered implicitly through conflict for finite resources. Different species find different combinations of environmental factors--different niches--which are relatively uninteresting to other species. Individuals of the same species use those resources until there is conflict. At that point, they vie for the same turf, food, and other environmental resources, and the increased competition and conflict cause individuals of the same species to share with one another, not out of altruism, but because the resources they give up are not worth the cost of the fight. It might be possible to induce similar conflict for resources in genetic optimization. Unfortunately, in many optimization problems, there is no natural definition of a resource. As a result, we must invent some way of imposing niche and speciation on strings based on some measure of their distance from each other. We do just this with what we have called a sharing function.

A sharing function is nothing more than a way of determining the degradation of an individual's payoff due to a neighbor at some distance as measured in some similarity space. Mathematically, we introduce a convenient metric d over our decoded parameters x_i (the decoded parameters are themselves functions of the strings $x_i = x_i(s_i)$):

$$d_{ij} = d(x_i, x_j)$$

Alternatively, we may introduce a metric over the strings directly:

$$d_{ij} = d(s_i, s_j)$$

In this paper, we use a metric defined over the decoded parameters x_i (phenotypic sharing); later on, we briefly consider the use of metrics defined over the strings (genotypic sharing). However we choose a metric, we define a sharing function sh as a function of the metric value sh = sh(d) with the following three properties:

1. $0 \le sh(d) \le 1$, for all d
2. $sh(0) = 1$
3. $\lim_{d \to \infty} sh(d) = 0$

Many sharing functions are possible. Power law functions are convenient:

$$sh(d) = 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha}, \quad d < \sigma_{share}$$
$$= 0, \quad \text{otherwise}$$

In this equation, σ_{share} and α are constants, and Figure 4 displays power law sharing functions with α values equal to one, greater than one, and less than one.

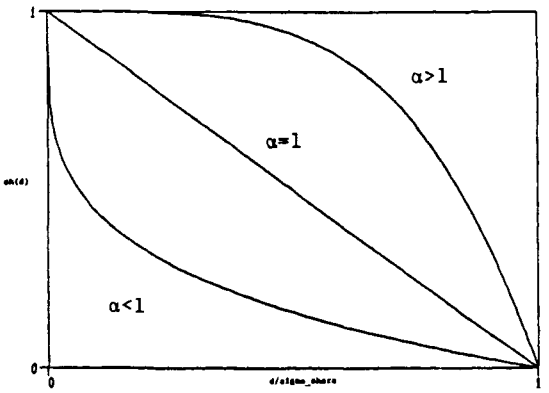


Figure 4. Power law sharing functions sh=sh(d).

Once we have selected a metric and a sharing function, it is a simple matter to determine a string's realized or shared fitness. We define a string's shared fitness f' as its potential fitness divided by its niche count m_i':

$$f_i' = \frac{f_i}{m_i'}$$

The niche count m_i' for a particular string i is taken as the sum of all share function values taken over the entire population:

$$m_i' = \sum_{j=1}^{N} sh(d_{ij}) = \sum_{j=1}^{N} sh(d(x_i, x_j))$$

Note that the sum includes the string itself. Thus, if a string is all by itself in its own niche ($m_i' = 1$), it receives its full potential fitness value. Otherwise the sharing function derates fitness according to the number and closeness of neighboring points.

RESULTS

We evaluate the use of sharing functions through computational experiments on two multimodal problems. The first function is a periodic function with five peaks of equal magnitude:

$$f_1(x) = \sin^6(5.1\pi x+.5)$$

We compare the performance of a simple genetic algorithm with stochastic remainder selection, crossover, no mutation, and no sharing to the performance of the same GA with sharing. We use a triangular sharing function (α=1) with σ_{share} = 0.1; the metric d is taken as the absolute value of the difference between the string x values. Length 30 binary strings are decoded as unsigned binary integers and normalized by the constant 2^{30}-1. Genetic algorithm parameters have been held constant across all runs as follows:

Probability of mutation p_m = 0.0

Probability of crossover p_c = 0.8

Population size = 50

Maximum number of generations = 100

We seek methods that maintain appropriate diversity without introducing arbitrary diversity through mutation or other means. Therefore, we have set the mutation probability to zero to put the sharing function technique to its most stringent test: if appropriate diversity can be maintained without mutation, we should expected similar or better (off-line) performance when mutation is present.

Runs with and without sharing are started from the same population generated uniformly at random. After 50 generations the run without sharing has lost all points at two peaks as shown in Figure 5. By contrast, the run with sharing has stable subpopulations roughly equal in size at all five peaks as shown in Figure 6. Similar graphs at generation 100 are shown in Figures 7 and 8. Note how the run without sharing (Figure 7) has completely converged to a single peak even though there is no selective advantage for any peak. By contrast, the run with sharing remains committed to stable subpopulations clustered about each peak. This latter result is especially remarkable considering that no mutation has been used. With no mutation, once an allele is lost at a particular locus, there is no way to get it back. The existence of stable subpopulations about each peak shows how the sharing function maintains appropriate diversity--the necessary, sometimes competing building blocks--required to exploit all five peaks.

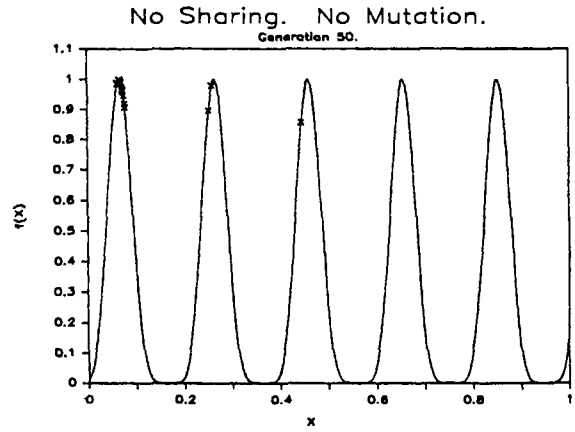


Figure 5. GA without sharing concentrates points at three peaks after 50 generations on function f_1 (equal peaks).

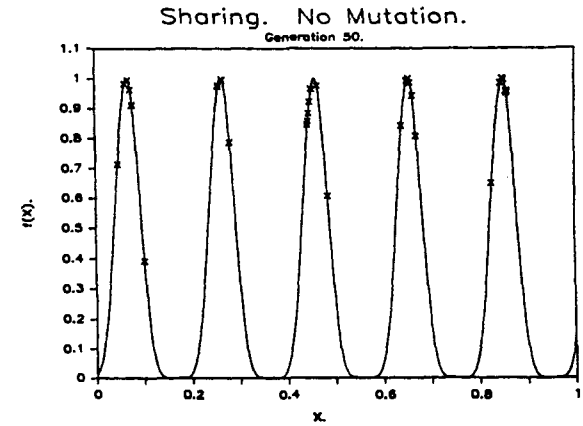


Figure 6. GA with sharing distributes points to all five peaks after 50 generations on function f_1 (equal peaks).

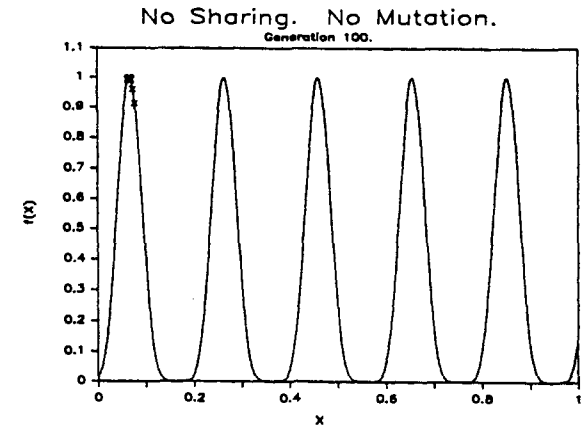


Figure 7. GA without sharing concentrates all points on a single peak after 100 generations on function f_1 (equal peaks).

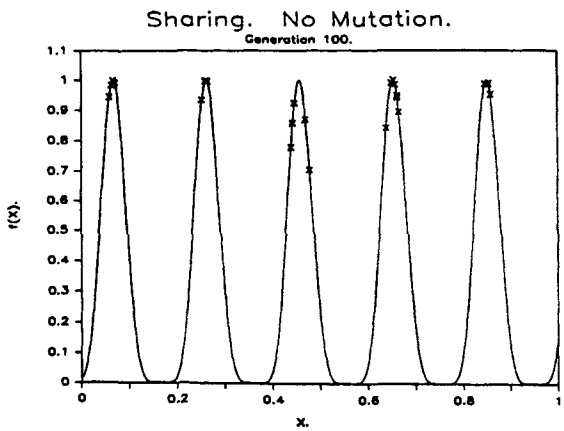


Figure 8. GA with sharing continues to distribute points among all five peaks after 100 generations on function f_1 (equal peaks).

The second test function $f_2(x)$ has five peaks with decreasing peak magnitude as given by the following equation:

$$f_2(x) = f_1(x) \cdot e^{\left[-4\ln 2 \frac{(x-0.0667)^2}{0.8^2} \right]}$$

We make comparisons of the two GAs, with and without sharing, using the same parameters and string coding as before. Specifically, we compare the two cases at generation 100. By that time, the genetic algorithm without sharing has allocated all of its trials to the highest peak as shown in Figure 9. By contrast, the GA with sharing forms stable clusters of points about four of the five highest peaks with cluster size roughly proportional to peak fitness as shown in Figure 10. This is the kind of performance we predicted earlier, except for the lack of strings at the lowest peak. To understand why this has occurred we briefly return to the theory of sharing presented earlier.

From our earlier discussion, we expect a stable equilibrium to form when the following equations hold true:

$$\frac{f_i}{m_i'} = \frac{f_{i+1}}{m_{i+1}'}$$

for all niches i, and

$$\sum_{i=1}^{M} m_i' = N$$

where M is the number of niches and N is the population size. It may be shown that this set of equations predicts proportions of niche members as the ratio of niche fitness to the total fitness. On function f_2, we expect a total number of trials to be allocated to the lowest peak as follows:

$$50*0.075/(0.075+0.22+0.5+0.85+1.0) = 1.42$$

In other words, we expect approximately one individual to remain at that peak, and we should not be surprised that no strings cluster there. If we want to maintain a subpopulation at such a low peak, our theory suggests that we need a larger population to overcome the unavoidable errors of stochastic sampling and selection.

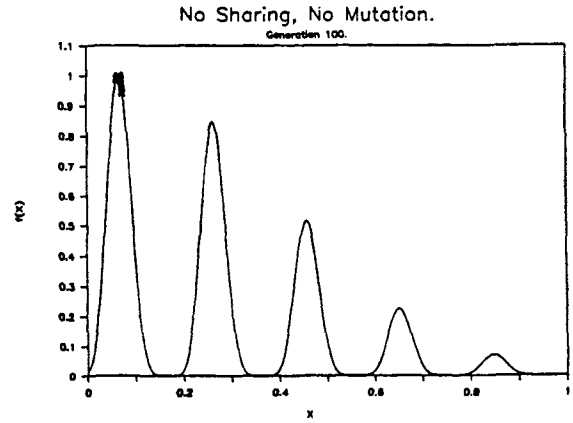


Figure 9. GA without sharing concentrates all points on highest peak (at generation 100) on function f_2 (decreasing peaks).

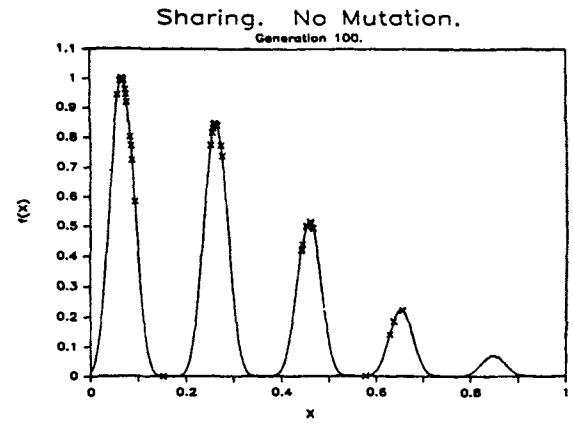


Figure 10. GA with sharing distributes points to all but lowest peak (at generation 100) on function f_2 (decreasing peaks). Lowest peak has expected population size of only one member, not enough to overcome selection and sampling errors.

EXTENSIONS

The method of sharing functions is not limited to one-dimensional problems. Sharing functions may be evaluated using any reasonable metric that includes any number of problem parameters. Additionally, there is no reason to limit the method to phenotypic sharing (where the measures are calculated based on differences in the phenotype--the decoded problem parameters). As an alternative, sharing functions may be evaluated using the genotype (the string) directly. In this form, the Hamming distance (the number of different bits) may be an especially useful metric.

In many cases, there may be no need to perform the sharing calculations as precisely as has been implied by the above equations. With the full formulation, $\binom{N}{2}$ sharing function evaluations are required (N^2 if symmetry is not exploited) to calculate the niche count values m_i exactly. This level of computation may be reduced by taking a random sample of k (k<<N) share function values and extrapolating the mean. If the mean of the k share function evaluations for string i is μ_i and the population is of size N, then the following formula provides a reasonable way to estimate the niche count m_i':

$$m_i' = (N-1)\mu_i + 1$$

Although this approximate niche count method has not been tested, Monte Carlo sampling techniques have been adopted in at least one other GA study with success (Grefenstette & Fitzpatrick, 1985). There is every reason to suspect that cheaper, approximate niche count estimates may be used without excessive performance degradation.

CONCLUSION

In this paper, we have developed a method for improving the performance of genetic algorithms in multimodal function optimization problems. This method uses sharing functions to induce artificial analogs to the natural concepts of niche and species, thereby permitting the formation of stable, non-competing subpopulations of points surrounding important peaks in the search space. The method has been tested on two multimodal functions, one with peaks of equal size and one with peaks of decreasing size. In both cases, the genetic algorithm with sharing is able to maintain stable subpopulations about significant peaks while an identical GA without sharing is unable to maintain points at more than a single peak. Additionally, the GA with sharing is also able to maintain stable subpopulations of appropriate size: the number of points in each cluster is roughly proportional to the peak fitness value. This automatic allocation of resources in a reasonable fashion should not only be transferable to other multimodal optimization problems, it should help in maintaining appropriate diversity in genetic algorithms without resorting to mutation and mutation-like operators that unnecessarily degrade on-line performance. These proof-of-principle results should permit the extension of these methods to other larger and more complex problems of genetic optimization.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant MSM-8451610.

REFERENCES

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In J. J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and Their Applications (pp. 101-111). Pittsburgh: Carnegie-Mellon University.

Booker, L. B. (1982). Intelligent behavior as an adaptation to the task environment. (Doctoral dissertation, Technical Report No. 243. Ann Arbor: University of Michigan, Logic of Computers Group). Dissertations Abstracts International, 43(2), 469B. (University Microfilms No. 8214966)

Cavicchio, D. J. (1970). Adaptive search using simulated evolution. Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 36(10), 5140B. (University Microfilms No. 76-9381)

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 44(10), 3174B. (University Microfilms No. 8402282)

Goldberg, D. E. (1986). Simple genetic algorithms and the minimal deceptive problem (TCGA Report No. 86003). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., & Thomas, A. L. (1986). Genetic algorithms: A bibliography (TCGA Report No. 86001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, SMC-16(1), 122-128.

Grefenstette, J. J., & Fitzpatrick, J. M. (1985). Genetic search with approximate function evaluations. In J. J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and Their Applications (pp. 112-120). Pittsburgh: Carnegie-Mellon University.

Grosso, P. B. (1985). Computer simulation of genetic adaptation: Parallel subcomponent interaction in a multilocus model. (Doctoral dissertation, University of Michigan, University Microfilms No. 8520908).

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

Mauldin, M. L. (1984). Maintaining diversity in genetic search. Proceedings of the National Conference on Artificial Intelligence, 247-250.

Perry, Z. A. (1984). Experimental study of speciation in ecological niche theory using genetic algorithms. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 45(12), 3870B. (University Microfilms No. 8502912)

Shaffer, J. D. (1984). Some experiments in machine learning using vector evaluated genetic algorithms. Unpublished doctoral dissertation, Vanderbilt University, Nashville.

Wilson, S. W. (1986). Classifier system learning of a boolean function (Research Memo RIS-27r). Cambridge: Rowland Institute for Science.

References

THURSDAY, JULY 30,1987 6:30 - 10:00 CONFERENCE BANQUET: New England Clambake crossover operators on the traveling salesman problem classifier based system for discovering scheduling heuristics algorithm to generate LISP source code to solve the prisoner's dilemma Optimal determination of user-oriented clusters: an application for the reproductive plan algorithm and biological development algorithms and communication link speed design: theoretical considerations algorithms and communication link speed design: constraints and operators

bound on the expected number of representatives of a particular schema (similarity subset) in the next generation under various genetic operators. In this

paper, assuming a large population of binary,

haploid structures of known distribution, processed

by fitness proportionate reproduction, random mating, and random, single-point crossover, an exact expression for the expected proportion of a

particular string (or representatives of a particular schema) in the next generation is calculated. This derivation is useful in analyzing the expected performance of simple GAs. genetic algorithm on a given problem with specified coding; they may be used for calculating the correct expected propagation of a set of competing schemata; they may also be used for estimating disruption or source probabilities for particular strings or particular schemata in a specified population of structures. In the remainder of the paper, we develop our extended analysis in three steps. We reexamine the fundamental theorem of GAs (the schema theorem), calculate an exact expression for the probability of disruption due to crossover, and calculate the expected gain of individuals due to mating and crossover by others. THE FUNDAMENTAL THEOREM OF GENETIC ALGORITHMS INTRODUCTION Over the past two decades, the application of genetic algorithms (GAs) to search and machine learning problems in science, commerce, and engineering has been been made possible by a number of theoretical developments (Holland, 1973, 1975; De Jong, 1975; Bethke, 1981). Without these

theories, it is doubtful that many of us would have

made much sense of our computer simulations or

experiments; this speculation is supported by the

experience of genetic algorithm prehistory. We

need only recall some of the evolutionary schemes that
resorted to mutation-plus-save-the-best

strategies (Box, 1957; Bledsoe, 1959; Friedman,

1959; Fogel, Owens, & Walsh, 1966) to remember that

shots in a darkness without schemata or the

fundamental theorem (Holland, 1975) can be

frustrating experiences indeed. Because of the

usefulness of theory to progress in genetic

algorithm research, there is still a pressing need

to improve our understanding of the foundations-

the theoretical underpinnings--of genetic

algorithms and their derivatives. In this paper, we extend
the fundamental

theorem of genetic algorithms to exactitude.

Specifically, we derive a complete set of equations

describing the combined effects of reproduction and

crossover on a large population of binary, haploid

structures. These equations may be used to

EXPECTED

VALUE 1.0 2.0 - MAX A B 1 N/2 INDIVIDUAL A MAX
1.0 N 2 0 100 95 F F 9 0 85 80 75 1.2 1.4 1.6
1.8 M A X Value Figure 2 - Observed F F s 2 RSSwoR
RSIS .SUS=theoretical o p t i m u m E R R O R IN F F s
4 3 RSSwoR 2 1 0 1.2 RSIS SUS 1.4 1.6 M A X
Value Figure 3 B i a s Severity 1.8 2 2 1 RSSwoR 2.0

1.5 Biaa Factor inapplicable region RSIS o p t i m u m 1 sus 0.02 0.06 0.1 Expected Va lue Figure 4 - Bias Direction (Opt imal Bias Factor = 1.0) 0.9 0.94 Expected Value 0.98 RSSwoR 1 RSIS SUS Bias Factor 1.5 2 A = ev : 0.000,RSSwoR B = ev: 0:001,RsswoR A = ev: 0.009,RSIS B = ev: 0.091, RSIS 6 4 B I A S

but only recognizes about three of the patterns. With crowdsample = 10, as in the second run de-

scribed, it was found that the recognizer population

maintains a high number of schemata at all times, keep-ing the number of recognizer schemata around 100 ± 25

each time step. By maintaining such diversity in the rec-ognizer population, all pattern schemata of 5 or fewer defining bits are found by the genetic algorithm along with a few schemata with even more defining bits. This is also due to the recognizerpopulationsize parame-ter. Theory predicts that when there are 2 n schemata of length n, a population of size 2 5 = 32 or greater should be able to search these. A population of size 50, where 2 5 < 50 < 2 6 , does this plus finds a few other longer schemata. One of these longer schemata, not in the initial recognizer population, has been created by putting together two shorter schemata that increase in frequency enough to form the longer schema, which then increases in frequency. The shorter schemata de-crease in frequency since they have been incoporated into the longer schema. Thus, we see the beginning of a default hierarchy forming[7][9]. First shorter, more gen-eral schemata are tested against the environment, and then the useful ones are used to build longer, more spe-cific schemata that predict the environment (or in this case cover the strings in the pattern population even more accurately). With a larger population, more of the longer schemata should emerge. 5 Conclusion The recognition and covering constraints are the mini-mal ones that the current model can satisfy. It does this by taking two shorter schemata and applying crossover to the strings of those schemata to form a new longer schema. This longer schema is a combination of the two shorter ones and is in a string that matches a pattern at more bits. Thus, a default hierarchy is emerging in the recognizer population. Mutation also can create such longer schemata and fitter strings but only one more bit will match an pattern. The one additional matching bit does not increase the fitness of the string nor the number of recognizer strings to match pattern strings as much as recombination does. However, these results are based on

preliminary runs and need to be quantified more precisely before more specific conclusions can be made. There are still many questions to study about this pattern recognition model. 1. How is the default hierarchy of schemata repre-sented in the recognizer string population? Are there recognizer strings that contain several shorter pattern schemata and some recognizer strings with longer schemata? In terms of knowledge representation[6], is the representation fine-grained with longer, more specific schemata or coarse-grained with more gen-eral schemata? Is the entire emerging default hi-erarchy maintained in the recognizer population? 2. Do distinct subpopulations evolve in the recog-nizer population with respect to the patterns or other criteria such as regularities in the pattern strings? Does each recognizer string contain schemata for one pattern or for more than one pattern? If the patternpopulationsize is increased, will the system no longer be able to pick out all of the regularities which may no longer come from one pattern but may include several patterns? How will this depend on the recognizerpopulationsize? 3. How do the schemata in the recognizer popula-tion change when the initial pattern population is changed? And if one of the parameter values is changed? What if the fitness function is changed to give maximum fitness value when T bits match? 4. Are matchsample and crowdsample, which main-tain diversity in the recognizer population, also re-sulting in suboptimal fitnesses for the recognizers that cover the patterns? That is, does it become impossible for perfect matches to evolve for each pattern? Is this somehow an adaptive advantage even if it is not a performance advantage? How can their values be set by some feedback mecha-nism based on regularities found and the resulting fitness, so that the recognizer population can con-tinue to recognize and cover a pattern population that is changing its size and strings over time? 5. Is the final recognizer population robust to changes in the pattern population? Will the recognizer population fitness decrease alot quickly if an orig-inal pattern is replaced with a new and different pattern string? Will the entire recognizer popula-tion have to re-evolve to redistribute the pattern schemata it has, lose the ones no longer in the pattern population, and acquire new ones? Various tools still need to be built to analyze the evolution of schemata and the default hierarchy in the recognizer population. For instance, since the schema finding algorithm only finds a sample of the schemata present in the pattern population, a reasonably-sized phylogenetic tree can be plotted over time based on what pattern schemata in the recognizer strings give rise to which new and possibly longer pattern

schemata in new recognizer strings. The frequency of each pat-tern schemata in the recognizer population (equal to the number of recognizer strings with that pattern schemata) could be plotted over time and used with the phyloge-netic tree to trace the development of the default hier-archy. 33 where schema s has an average expected fitness f s (t) at time t, M s (t) is the number of strings in the recog nizer population that are elements of schema s, M is the total number of strings in the fixed sized recognizer population, and ε, is an error term created by having a schema instance get broken up by a crossover point or mutation. The size of the recognizer population and the sample sizes also affect the number of schemata and the number of their instances that are possible to have in the recognizer population. Thus, the schemata dynam ics can be studied under different parameter settings for the population and sample sizes, and the affects of fluc tuations occuring when new schemata are introduced into the recognizer population by crossover, mutation, or changes in the pattern population can also be stud ied. Once the structure and evolution of the default hi erarchy is better understood, this pattern recogniton model can be used to study the structural evolution of memory in other physical and biological systems that do pattern recognition. The features of the particular system can be assigned to the bits of the pattern and recognizer strings. Then one can study how the rec ognizer strings of that system's memory are structured with respect to the regularities in the patterns the sys tem is trying to recognize. For instance, in modeling the immune system, ex perimental evidence can be used to set up the fitness function and parameter values to be more realistic with real immune systems and to see if the model results in antibodies with similar characteristics as those found in the experiments. Some experimental studies have shown that one antibody can recognize more than one antigen (called "multi-specificity" in immunology) [13]. Such an antibody is a generalist. Other evidence shows that when new antigens are introduced, new antibod ies are created by an increased rate of point mutations. The affinity of these new antibodies for new antigens in creases ten-fold over a short period of time, indicating that the antibodies become very specific at recognizing the antigens [14]. Thus, some sort of default hierarchy is developing in the immune system. What is the na ture of this default hierarchy if the fitness function and paramters are set to correspond to actual numbers used and seen in these experiments? What antibodies result if the intial antibodies are ones that have evolved in this

pattern recognition model and can only undergo muta tion in the future when presented with a new antigen? How might a network of such antibodies compensate for any shortcomings of the antibody population which can no longer undergo crossover? Also, in a letter recognition system, letter patterns can be represented with line features of different lengths, angles, and positions. Could this model then generate a default hierarchy to recognize the letter or letters? How could it recognize the same letters if they are written in a different style? In cognitive science, a concept is represented by a prototype developed from a set of instances. The na ture of the prototype with respect to the properties of the instances is not yet clearly understood, but the in stances do not necessarily all share some set of property values which might then be the prototype. Rather, each instance shares one or more property values with one or more other instances in the set [11]. Translated into the pattern recognition model, the recognizer popula tion can try to match the instances in the environment where the prototype would appear as the more general schema that persists over time and is not just used to build one specific schema. On the other hand, the pro totype might turn out to be some piece of the default hi erarchy distributed over several recognizer strings. How would this correspond to any experimental data that has been collected by psychologists on the nature of proto types? What kinds of prototypes are actually commu nicated in these experiments (i.e. trasmitted verbally to the experimenter)? By understanding the organization of the default hi erarchy in the recognizer strings, the memories of other pattern recognition systems evolving under a genetic al gorithm can be analyzed as well. A system that tries to recognize patterns in the environment and then builds an internal model or memory of those patterns must use some definition of similarity to cluster the patterns into classes. If this system also transmits its internal model by a code representing how to reconstruct that model, then this system is a good candidate to be cast in and analyzed by this pattern recognition model. 6 Acknowledgements This work has been performed under the auspices of the U.S. Department of Energy. I would like to thank John H. Holland for his guidance and ideas about ge netic algorithms and modelling, Chris Langton for his comments on modelling artificial life, and Alan Perelson, Doyne Farmer, and Norman Packard for their valuable insights into the structure and dynamics of biological and physical systems. 34 The frequencies of the schemata can also be studied with analytical tools developed for understanding dy

namical systems. Every schemata changes its frequency (equal to the number of strings in the population be longing to that schema) in the population with respect to the probability P,(t + l) = M s (t +1) M = f s (t).ε s .M s (t)

A(M(.005).M''(2,.01),T(15,.5<

ABSTRACT The genetic algorithm has been demonstrated as an effective learning and discovery technique for a large class of problems. Despite its promise, however, the algorithm has not found wide acceptance among members of the artificial intelligence community at large. Our contention is that this is due to the representations used in genetic algorithm research. These representations are extremely simple, while much of the progress in artificial intelligence has been supported by highly expressive, relatively complex representations. In this paper we discuss the the adaptation of the genetic operators to a representation that captures the salient features of the description and processing of knowledge typical of high-level knowledge representation systems.

1. Introduction

Learning is any process by which a system improves its performance through modification of its behavior. The capacity for learning is a key element of intelligent behavior in living organisms, yet one that is almost totally absent in present intelligent artifacts such as computer programs. However, as we attempt to build increas-

ingly sophisticated problem-solving behavior into computers, the flexibility and adaptability afforded by learning approaches to problem-solving will become increasingly essential for success.

The genetic algorithms (GA) community has been engaged for well over a decade in research in automated learning and discovery. 2,3 This work has led to a number of impressive successes in the solution of difficult problems. 4,5,6 The approach is particularly interesting in its use of two powerful metaphors by which computation is organized -- the economy metaphor in the bucket brigade algorithm, and the metaphor of natural selection and evolution in the genetic algorithm proper. Yet this approach has failed to make significant inroads to the artificial intelligence (AI) community. We believe this is primarily due to three closely related factors: (1) the need to simulate very large numbers of generations in the GA to reach convergence on good problem solutions; (2) the need for great speed in the simulation in order to produce the numbers of generations needed in a reasonable time; and (3) the representation of the basic units of the GA as vectors, typically bit strings so that a large proportion of the possible solutions can be represented by any population of strings.

AI is characterized in large part by its attention to highly expressive knowledge representations. The theoretical advantage of the bit-string representation is disavowed in most AI research programs. Many programs depend on logically tagged data structures that are intended to be interpreted directly as human cognitive models of a domain. The mechanisms which exploit those knowledge representations tend to be adequately fast for decision aids and other interactive programs, but are very much slower than the speeds demanded of GA algorithms in the past. However, the speed at which learning is achieved may not need to be as fast for knowledge-based systems as has been needed for GA research and previous GA applications. The GA typically starts with little knowledge of correct solutions, and does much of its work converging on good approximate solutions. In contrast, knowledge-based systems are knowledgeengineered precisely to embody good solutions at the outset of their operational deployment. Moreover, such systems are expected to have a fairly long operating life-span. Consequently, for many AI applications it is reasonable to trade the speed of the bit-string GA for a slower genetic processes based on the more complex representations found in typical knowledge-based systems. 2. Project Objectives and Organization of the Paper This paper describes work conducted as part of a machine learning project at MITRE. The overall objective of the project is to improve the performance capacity of knowledge-based systems by imbuing such systems with an ability to learn from their "experience". For an automated system to exhibit learning it must (at some level) be able to evaluate its performance to determine whether its behavior is adequate to its tasks, and it must be able to modify its behavior if not. For a knowledge-based system the modification, and therefore the evaluation, must occur at the knowledge level, since that is the level at which the system was designed to be open to modifica-tion. In the context of such a system, we interpret "experience" to mean 1) that which the system is told by the knowledge engineer, 2) the system's input data stream, and 3) the intro-spective examination of the knowledge base, data base, and system results by the system itself. In 69 such knowledge-based systems, the bucket brigade is appropriate as an evaluation mechanism of system performance, the GA as a discovery algorithm for system improvements over the long term of the life of the system. We have explored the utility of the bucket brigade algorithm for knowledge evaluation in traditional rule-based application systems 7 and for one variant of rule-based inference under uncertainty. 3 The principle problem which we are

addressing in this paper is the generation of legal rules within a high-level rule representation language. We reconsider the GA in light of representations widely used by the mainstream AI community. In the past the GA has been applied to high-level problems by developing a representation for the problem that can be mapped to an underlying GA vector represen-tation. 6,9,10 The mapping has followed two paths, often called the "Michigan" and "Pittsburg" approaches to the GA after the schools at which their respective research teams established each approach. The Michigan approach views the GA as working as a process internal to a system, so both evaluation and modification occur in the context of a simple system adaption, to its environment. The Pittsburgh approach views each represented individual in the GA as an entire system, for instance a possible rule-base, which is in compet-ition with other systems to establish itself in an environment. 12 Our work can best be understood as that part of the "Michigan" tradition concerned with mapping GA approaches to complex representa-tions internal to a system. The GA has in the past been applied to existing high-level representations indirectly, through the creation of mapping functions to interpret GA results at the representation level. We are engaged by the other side of the problem. Given a knowledge representation, we are asking how we might modify the genetic operators to accommo-date that representation. We believe the representation we have chosen is sufficiently general to be of interest to a wide audience. We hope that, in turn, the computing paradigm repre-sented by the GA will be fruitfully applied within the traditional computing approaches in AI, and in increasing numbers of real-world AI applications. Finally, we hope to clarify the relationship and relative utility of GA approaches to other learning and discovery techniques in the field. The rest of the paper is organized as follows. Section 3.0 describes our representation. In Section 4.0, we reconsider the operators that form the heart of the GA from the point of view of the representations of Section 3.0. We point out some issues raised by this work in Section 4.0, and conclude the section and the paper with a description of the status of this work and the expected activities in this area for the next year. 3.0 The Knowledge Representation We represent knowledge in two forms: the "static" knowledge and vocabulary of the system, in our case an object description and datatype language expressed as a taxonomy of the terms of interest, and the "procedural", dynamic knowledge of the system expressed by pattern-invoked conditionaction production rules. Our description of these forms will be a slight

simplification of the actual representations in our system. The simplifications are aimed at avoiding the difficulties of explain-ing aspects of production systems that don't bear directly on the adoption of GA operators to the representation. The conceptual content of the system is expressed at four levels corresponding to the components of the rule language. They are objects and their characteristics or attributes, such as object 1 and (location of object 1); relations among objects and their characteristics, expressed in clauses such as (i s -wi th in-d i s tance (location of object 1) (location of object 2) distance) ; conditionals, which are the relations on those clauses that constitute rule left-hand sides, e.g., (AND c l a u s e 1 clause 2); and the rules themselves, which express inferential relations with conditionals as antecedents, such as IF (AND clause 1 clause 2) THEN assert (clause 2) . 3.1 Objects The objects are organized in a taxonomy of types of objects, thus every object can be characterized by its class membership as well as the properties attached to it. Properties of a class* are inher-ited by all members of the class. An example object definition hierarchy appears in Figure 1. Figure 1 Concept representation: Objects Univcroe_of_dincouroe relation oioum.nl-t.poi o b j e c t location vehicle ir».d building d a t a t y p e location distance vftthtn-diat |loc IM di . l) •"• veh 1 loc=1 apd=1 veh 2 loc=2 apd=2 bldg 1 loc 1+1 aise 1+1 bldg 2 lot 2+2; aise 2+2 The leaves of the tree are instantiated objects, each of which has a set of characteristics as specified by the class of objects to which it belongs. The leaves of the tree are instantiated objects, each of which has a set of characteristics as specified by the class of objects to which it belongs. 3.2 Relations Relations are expressed in prefix notation, with the relation name followed by an ordered list of arguments wrapped in parenthesis. The arguments are typed, the available datatypes being provided by the concepts defined at the time the relation is defined. Relations and datatypes are represented * The collection of properties of a class are the necessary and sufficient conditions for membership in that class. 70

as terms in the concept language in the same way

that objects are, with properties and class member-

ship. A relation may be expressed among constants

or variables of the appropriate types, with the use

of variables in relations providing quantification. An instantiated relation, one with all constants in its arguments, takes on a value (true or false for predicates, a member of the function range for functions). Figure 2 illustrates a relational form, a relation with free variables, and an instantiated relation.

Figure 2

Concept representation; Relations (Relation 1 concept-type 1 concept-type 2 ...) (within.distance ?location 1 ?location 2 ?distance) (unthtn Pittance (location vehicle 1) (location uekiele 2)10ft)

3.3 Conditionals

The left-hand side of a rule expresses a predicate relation among a set of propositional clauses. The predicate relation forms the condition under which the rule can be applied. There are three types of propositional clause: short term memory elements (STM), associations, and predicates. We give these special attention because of their roles in the left-hand side of inference rules. STM are used to introduce an arbitrary number of new variables into the conditional part of a production rule. Facts that are expressed to the system in such a way as to legally bind to (unify with) all the variables of a STM pattern are said to match that pattern. Variables must be bound in the datatype of the

corresponding relation argument. Variables may

also be constrained to be any subtype of the

relation argument's datatype, as defined in the

concept hierarchy. Variables bound to values in

one STM must be bound in the same way whenever the

variable is mentioned in another proposition of the

left-hand side.* Associations introduce a single new variable

into the left-hand side by binding that variable to

the value of a function call. The function's

arguments must all be constants or previously bound

variables. Association clauses function to set up

variable bindings for the clauses.

Predicates are the only other type of clause

directly expressed in the left-hand side. They

introduce no new variables. As with functions in

association clauses, all the predicate's arguments

must be constants or bound variables. The propositions of the conditional part of a rule are considered in some relation to each other, and the values of the propositions combined according to that relation. The combination operator collects and combines the values of the clauses of the left-hand side. For simplicity we consider only combination operators "and" and "or". "Not" is viewed as a relation that may appear as a proposition in the conditions (where "and" and "or" may also appear, recursively). Any clauses that mention variables bind those variables to appro-priate values. Bindings remain in effect over the entire scope of the rule. Figure 3 illustrates one possible form of a conditional. Figure 3 Concept representation: Relations in rule left-hand sides.
(combination-operator (STM-rtlalion 1 ?var 1 constant 1
) (STM-retatton 2 ?var 2 constant 2 constant 2)

(Associate ?var 3 (function 1 ?var 2 constant 4)) (predicate 1 ?var 3 ?var 2)) 3.4 Production Rules Figure 4 illustrates a typical rule in our system. The rule's conditional clause contains the combination operator "And", indicating that all participating clauses must be true for this rule to be a candidate for firing. Each participating clause in the conditional denotes a relation whose value plays a part in the combination method. (Associate clauses are not participating clauses in these relations). The matching of the rule's left-hand side makes it a candidate for taking its right-hand side action and it is added to the conflict set. Figure 4 Concept representation: rules. IF (A N D (Ret 1 ?varl const 1) (Rel 2 ?varl const 2)) T H E N (D O (Action t ?var l)) The actions a rule can take fall into three categories, adding STM elements to working memory, deleting such elements from working memory, or evaluating an arbitrary function. No new variables are bound on the right hand side, and all variables mentioned are bound to the the corresponding values of their bindings on the left-hand side. For example, a constant C o n s t 3 bound to ?varl such that both (Rel 1 ?var l c o n s t 3) and (Rel 1 ?var l c o n s t 2) are true results in (Ac t ion 1 const 2) taking place.

∗ One might think of a list of possible bindings

being carried from left to right across the list of

propositions, such that at any break between

propositions one can construct the bindings so

far. We will use such a view in describing GA

operators in the representation in Section 4.0,

below. 71 3.5 Rule Grammar The system described should be naturally express-ible by a context-sensitive grammar, since each category is systematically well-formed and each depends on context information (e.g. variable bindings) provided by previous rule elements. Such a grammar would provide a formal framework for describing the construction of legal rules, includ-ing variable binding decisions, as derivation trees. We are preparing such a grammar to support a version of crossover for this rule represen-tation. Its role is described in section 4.5, below. 3.6 Control The minimal control structure for a production system consists of a three-step cycle, plus an initial action taken by a "prime mover" (typically the

user). The "mover" adds a working memory element to the database of facts on which the system operates, simulating what would otherwise be control cycle. In the second step the resultant set of candidate rules to fire, the conflict set, is computed. Step three consists of a rule being chosen and firing.* The rule may add or delete elements of working memory, initiating a new cycle. The conflict set is updated with the new set of matched rules, and so on. The system halts when the conflict set is empty at the end of a cycle. We have described a pattern-invoked, strongly datatyped knowledge representation. The represen-tation is typical of a well-organized AI applica-tion system based on such tools as OPS-like rule interpreters and FRL-like declarative representa-tion systems. In the next section we discuss the implications of adopting GA type learning and discovery algorithms to such systems. 4. Genetic Operators for Pattern-invoked, Datatyped Representations The genetic algorithm simulates evolution and natural selection over a population of individuals in an artificial environment. The population is usually represented as a set of simple production or rewrite rules such as the one illustrated in Figure 5. * If several rules are chosen for firing simultane-ously we have a parallel system. Since we are concerned, in this paper, with the manner of automated rule construction, and not (directly) with the results of inference using those rules, we needn't concern ourselves with the issues parallel-ism normally raises. Such issues may need to be raised in considering how to control the appli-cation of the GA operators, but, except for the comments on Section 5.0, that is beyond the scope of this paper. Figure 5 Bit-string representation of productions. |0 |0 |1 |# |1 |O |# |0 | |0 |0 |1 |1 |1 |0 |# |0 | Abstractly, the bit-string represents the possible states of the world. Each position of the string may be thought to correspond to an independent proposition about the world, so all strings of 0's and 1's correspond to particular possible world states. A "don't care" character, "#", is intro-duced to make it possible to refer to sets of possible world states. In the context of a bitstring production rule, a left-hand side string represents the (set of) state(s) in which the rule may rewrite the world-state as its right-hand side. For example, in Figure 5, a world-state of 00101010 would be rewritten by tne rule as 00111010. Below we consider how to express the operators developed in this representation in the language of relations described in the last section. We will first describe the operator of interest in terms of the bit-string manipulations it performs. We then consider mechanisms which carry out an analogous

operation in pattern-invoked, datatyped representa-tions. We do not consider modifications at the object level, which is considered the primitive level of the system -- analogous to bit positions and values in the bit-string representation. Nor do we consider the rule level, since considerations of the relation and conditional levels suffice and can be trivially extended to include the right-hand sides of rules (no new variables being introduced there). We don't mean to imply that modifications in the action parts of rules are uninteresting, just that the mechanics of those modifications can be understood completely analogously to modifi-cations of the left-hand sides of rules. The operators we consider are generalization, specialization, mutation, inversion, and cross-over. Because of the importance of hierarchical concept representations in many AI systems, the issues in concept specialization and generalization have been the concern of a large part of the machine learning community. We therefore treat them separately, even though specialization and generalization are, strictly speaking, sub-species of mutation. 4.1 Generalization Generalization is achieved in the bit-string representation by changing a "0" or "1" to a "don't care" symbol ("#"). The utilization of "#" within the condition of a bit-string production provides an opportunity for that production to apply to more world states. It is used as a global patternmatching variable. Figure 6 shows an example of a bit-string production being generalized by switch-ing the 0 in position 1 for a # in position 1. 72

Figure 6

General izatot of Rvle in Fiqurc 5.

Old rule: |o |o |1 |# |1 |o |# |o | |o |o |1 |1 |1 |o |# |o |

New rule: |# |o |1 |# |1 |o |# |o | |o |o |1 |1 |1 |o |# |o |

4.1.1 Generalization at relation level

Instantiated relations in the rule language can be

generalized completely analogously to the bit

string operation by changing a literal to a

variable. Due to the strong typing of our

relational language, constraints apply to those variables. A variable might be bound to any member of the class to which the literal belongs, or to any member of a superclass of the literal, up to the datatype of the relation argument that the literal was instantiating. For instance, the instantiatied relation (location-of bldg 1) can be generalized by replacing the reference to a specific building by reference to a class of buildings or objects, such as (location-of ?bldg) or (location-of ?object), but not Just any element of the universe of discourse. Notice that (location-of ?vehicle) is not a generalization of, e.g., (location-of ?building).

A very similar mechanism can be brought to bear to affect generalization at a slightly higher level. The relation language itself is defined in a taxonomy in which a particular relation may have parent relations of which they are specializations. Such parent relations would have the same or fewer parameters of the same or more general types. Generalizing a relation directly would require, at most, removing one or more arguments from the previous relation's argument list.

4.1.2 Generalization of conditionals and

productions Generalization can also be achieved by removing clauses from the conditional, thereby removing a requirement for invoking the right-hand side action. Removal of an entire clause has to be handled with care because of the interaction of variable binding among conditional clauses and the actions of the rule. If a clause introduces a set variables must be purged from the rule. The easiest way to do this is to simply remove all replace such variables with variables of the same class introduced elsewhere in the conditional. So dropping the first clause of the conditional in Figure 3 would necessitate either the removal of clause 3 and then clause 4(!), or it would require changing ?var 1 in clause 3 to, e.g., ?var 2 from clause 2. On the other hand, dropping clause 4 could be done without further adjustments because it introduces no new variables. If we keep to the analogy that generalization of a literal or a variable is like "0" or "1" turned into "#", it's difficult to see what analogy to make of generalization at the relation level, unless it is something like turning some pattern of "0"'s and "1"'s into a pattern of "#"'s. The difficulty of finding a GA analog seems rooted in the richly layered representation inherent in the relation-based approach, whereas the simple description of the GA presented here is for only one level of process. Although it is possible to organize layered GA processes, it is a particular strength of most AI representations that they do so naturally. The price for such expressiveness, apparent here, is in the degree of complexity of operations on such representations. 4.2 Specialization The specialization operator in a bit-string

operation changes a "don't care" symbol to a literal value of 1 or 0. The example in figure 7 shows the fourth position of the condition specialized to a 1 value. This restricts the application of this rule to fewer possible world states. Figure 7 Sptctalization of Rute in Figure 5. Old rule: |o |o |1 |# |1 |o |# |o | |o |o |1 |1 |1 |o |# |o | New rule: |o |o |1 |1 |1 |o |# |o | |o |o |1 |1 |1 |o |# |o | 4.2.1 Specialization at relation level The mechanics of specialization in relations is very similar to generalization. Rules can be specialized by filling variable positions with literal values that have a type of the correspond-ing relation arguments (a constructor being used to generate a legal literal value based upon its type in the class hierarchy), by constraining the variable to be of a subtype of the type at which it is presently constrained, or by replacing the relation with a subspecie of the relation. 4.2.2 Specialization at conditional level Another method of specialization is the addition of clauses in the left-hand side of the rule. In the simple case the new clause does not introduce any new elements, it only further constrains the conditions of the rule's applica-bility. However, when new variables are introduced 73 by a STM or associate clause, the clause will do little useful work unless at least one of the newly introduced variables is mentioned elsewhere in the rule.* Therefore, specialization by clause introduction needs to be tied to the existing clauses through an existing variable or must be complemented by another process other clauses to effect such a tie. 4.3 Mutation Mutation is very similar to specialization and generalization in that it is conceptually a local operator that transforms individual elements of a rule construct. In the bit-string representation this transformation consists of replacing a "1" with a "0" and vice versa, see figure 8. Figure 8 Muation of Rule in Figure 5. Old rule: |o |o |l |# |l |o |# |o |- |- |o |o |l |l |l |o |# |o | |l |o |l |# |l |o |# |o |o |o |l |l |l |o |# |o | 4.3.1 Mutation at relation level In the production rule representation, mutation is constrained by the same datatyping considerations as specialization and generalization. It produces value changes based upon the type of the argument corresponding to the selected position, but, as opposed to those two operators, it seeks sibling concepts for the new variable or constant. Recall the instantiatied relation (location-of bldg 2) that was generalized by replacing the reference to a specific building by reference to a superclass of the building. A mutation on that clause would be, for instance, (location-of ?vehicle), which, as was pointed out in the discussion of generalization above, is not a

generalization of the instantiated relation. 4.3.2 Mutation at conditional level Mutation of the relation expressed by the clause can be viewed as the removal of a clause from the conditional followed by the addition of a new clause. Notice that in a fully articulated concept graph, the relation expressed by the new clause must be, at some level, a sibling or cousin relation to the original. One might use this to provide a metric and control regimen for the degree of mutation in a rule . *This is by no means always the case. Sometimes variables are introduced to test for some general condition to decide if an action should be taken, resulting in a rule like IF (STM ?var 2) THEN DO taction constant 1). 4.4 Inversion The inversion operator of the GA produces an end for end reversal of positions in the condition of a rule. It can be applied as a mutation operator during the running of the system, or as a crossover/mutation operator applied during reproduction. The example in figure 9 shows the basic notion behind inversion. Figure 9 Inversion of rule in Figure 5. Old rule: |o |o |1 |# |1 |o |# |o | |o |o |1 |1 |1 |o |# |o | New rule: |o |# |o |1 |# |1 |o |o | |o |o |1 |1 |1 |o |# |o | 4.4.1 Inversion at relation level Because of the strict ordering of arguments within relations, inversion cannot be in general achieved unless the types of the relation's arguments are symmetrical. Under any other circumstances, inversion would result in illegally formed clauses. 4.4.2 Inversion at conditional level The production system as described restricts ordering of clauses according to the introduction of variables. Inversion of the clauses in the conditional would presumably destroy some of the dependencies of that variable introduction scheme, requiring renaming and rethreading of variables through the inverted clause set. Without an explicit relation between, say, the clauses in a rule and the actions to be taken in a plan, it's difficult to see the point of inversion. Given the variable set for a rule, it shouldn't make any semantic difference in what order the clauses are expressed. 4.5 Crossover The crossover operator is the workhorse modification operator of the GA. It is also an operator which doesn't have direct analog in the "traditional" machine learning literature. The example in figure 10 shows two rules being crossed over after they were chosen as mates. A locus (denoted by the cut) determines how much of Rulel and how much of Rule2 will be used. In this example, Rulel provides the first 3 positions of the new rule with Rule 2 providing the remaining positions. 4.5.1 Crossover at relation level It is not immediately obvious that crossover makes sense for relations. In fact, the argument typing requirements are often cited as the cause

for crossover's inapplicability in this domain. 74 New rule:

Figure 10

Crossover of rule in Figure 5 and another rule.

Rulel: |o |o |1 |# |1 |o |# |o | |o |o |1 |1 |1 |o |# |o |

Rule2: |1 |l | # I | # | # | 0 |l |o | |0 |0 |l |l |1 |0 |# |o | I c u t ! 1 s t h a l f of R u l e 1 , 2 n d h a l f o f R u l e 2

N e w r u l e : |o |o |1 |# |# |o |1 |o | |o |o |1 |1 |1 |o |# |o |

However, there is a way in which crossover might be applied (the locus of crossover chosen) even within strongly datatyped relations when two rules are being crossed. This requires two modifications of the system we have so far described: (1) the system should be described by same context-sensitive rule grammar that formally delineates what is legal at all the levels as described in Section 3.5, (2) all rules should be described by a datastructure that indicates the derivation tree through the grammar followed in the construction of the rule. A rule can be cut at any point, but each part of the rule must carry a copy of its (partial) derivation. Crossover can then be performed by any two rule fragments whose derivations complete each other. That is, the cut results not only in the splitting of a rule into two fragments, but the tagging of

each cut end of the fragment, the left-hand side

with the partial derivation tree down to the cut.

The right-hand side with the subtree starting at

the art. The unification of the derivations

produces the new rule. This process is obviously

more complex than the other operators considered

thus far and we are presently trying to specify a

grammar for our system that will form the basis of

the derivation path datastructure.

4.5.2 Crossover at conditional level Figure 11 illustrates a naive view of

crossover when the cut occurs between clauses in

the conditional. Rulel and Rule2 are two

hypothetical rule mates. Their combination via

crossover produces a rule that mentions a variable

in its action that is not well-founded; ?varC is

supposed to be bound to (function ?varB), but ?varB

is unbound in New Rule). Moreover, ?varA is also

unfounded, and neither ?varA nor ?varl are doing

any work that is obviously related to the action to

be taken. Our first implementation of crossover

attempted to achieve crossover for conditions, and

initial experience to test some intuitions. It

promoted modification only among rules with the

same goal or class of goals. A rule was naively

produced. It then went through one round of

generalization, in which variables of the same type

with different names were unified and one round of

the creation and use of lists of tree

structured production rules varying in

length and complexity. Actions, conditions

and operators are randomly chosen from

tables of possibilities. Use of the

techniques is facilitated by the notions

of time delay and dependency in performing

tests or observations and by accounting

for indeterminate test results. GENES, a general program,
can adapt to a

variety of systems by changing the

contents of the various

accomodate the domain of

problem and by substituting

appropriate cases. tables to a particular a library of
INTRODUCTION The genetic algorithm is an iterative

adaptive search technique which has been

applied successfully to learning systems

having large search spaces and to problems

which cannot easily be reduced to closed

form. Holland, who originated the notion

of computer constructs which mimic natural

adaptive mechanisms, noted that in order

to apply this technique, the subject to be

treated must be capable of being

represented by some data structure and the

solutions must be capable of being

evaluated and ranked (1,2). APPLICATION TO EXPERT SYSTEMS
Conventionally in genetic algorithm

implementations, the rules containing the

expertise have been expressed in the form

of bit strings of fixed length. This

approach works quite well in terms of the

ease of applying the genetic operators

mutations may turn a 0 to a 1 or to some

other symbol taken from a similar simple

alphabet; crossovers are implemented by

randomly choosing two crossover points

along the string and exchanging the information between those two points. However there are many problems which cannot be easily expressed in terms of a simple alphabet, and still be amenable to the genetic operators (see, e.g. work on the traveling salesman problem (3, 4)). The commonly used dual valued alphabet is especially troublesome for two reasons. First, since the number of possible actual interpretations is unlikely to be a power of two, some interpretations will be redundant, weighting the possible choices unevenly. Secondly, some mutations would be extremely unlikely, e.g. 000 would probably not mutate to 111. These problems are both eliminated by having each decisional element be an index into an appropriate table. Choices for each position and changes due to point mutations are then made merely by randomly chosing one of the possible indices into that table. Schaffer and Grefenstette (5), building upon the LS-1 learning system originated by Smith (6), used a data structure where each expert was not an individual rule but, rather, an entire set of rules, in order to deal with multiobjective learning. These rule sets were of fixed length, although all the rules might not necessarily be

used. But there may be insufficient flexibility in solving difficult classes of problems where the expert is composed of linear data structures of fixed length, since practicable computer expert systems are often presented as a system of non-linear rules (7). These rules tend to be of the form: IF condition THEN action [{and l or condition}] where the action may be either an attempt to obtain more information on the system (an action rule), or a terminal decision in the nature of deciding the answer to the presented problem (a decision rule). 77 CREATING AN EXPERT GENERATOR We created GENES, a program to develop expert systems. Each expert in our system consisted of a linked list of rules, each rule parsed syntactically as a tree structure. The actual number of rules in each linked list was randomly determined via an average length parameter passed to the program upon its initiation. The number of rules allowed was optionally bounded. A different program parameter determined the maximum number of nodes for each rule. The minimal possible value was one, this single node being an action node which would mean that some prescribed action (randomly chosen from a table of possible actions) would always be carried out. One type of action was the undertaking of a specific additional test or observation; the other possible action was a final determination as to the state of the system - e.g. if the expert was a prospector, then this might be that a given mineral exists at a certain location in recoverable concentrations. There was only one action node per rule. Additional nodes each represented a possible condition precedent to carrying out the action, linked by boolean operators (again, randomly chosen). Conditions precedent nodes are of the form: IF {test-result-yields RELATIONAL OPERATOR scalar quantity}. As an example, if the expert created were a chemist, a possible node might be "if melting point > 180.51 degrees". In creating (but not in evaluating) the expert, booleans, relational operators, possible tests, and possible range of results were all randomly chosen from a permissible domain of actions, conditions {IF (NOT ((C1 = 2) AND (C4 > 3)) OR (C2 >= 12)) THEN Al} -> {A5} -> {IF ((C5 = TRUE) OR NOT (C1 <> 17)) THEN Al} Figure 1 Statements within {} indicate a single set of rules Statements beginning with C are conditions Statements beginning with A are actions -> indicates the linking of rules which are evaluated in order and values appropriate to the area of expertise desired for the system. Figure 1 shows a possible rule set for an unspecified expert. APPLYING THE GENETIC OPERATORS TO THE EXPERT POPULATION Three types of genetic operators were applied to the population of experts mutation,

crossover and inversion. The tree structured nature of each rule called for some modifications to the way these operators normally deal with fixed length strings of bits. Mutation was performed as an operation upon a single rule. A number of varieties of mutation were possible and the type of mutation occurring in each given case was chosen randomly from the list of possibilities, as was the rule to be mutated and the node of the rule at which the mutation would occur. The simplest type of mutation was a point mutation. This involved the change of one operator to another (e.g., > becoming > =). It was also possible for an operand of either a condition or an action type to be exchanged with another (e.g., in a chemical setting boiling point becoming melting point or 450 degrees might become 200). A third possibility is exemplified by the exchange of an OR for an AND or the addition or deletion of a NOT in the boolean expression. The last mutation possible was the addition or deletion of a rule or a portion of a rule - the latter being equivalent to a randomly selected branch of the parse tree. The next simplest genetic operation was an order inversion. Each inversion was carried out on a single set of rules. This involved randomly choosing, via the MOD operator, two points along a rule set of given length, and then exchanging the order of rules that existed between the two points. This operation did not affect the rules themselves but, rather, the order in which they were evaluated. The possible effects of inversion were twofold. First, the firing of an earlier rule (that is conditions evaluating positively so that an action is taken) could cause later rules to evaluate differently. With a new ordering, given preconditions now may or may not exist. Second, due to rule evaluation halting once a decision rule fired, new rules might come under consideration or old ones might not now be reached. The most complex genetic operator used involved crossover. Crossover, in this system, occurred at points between rules, and involved exchanges of identical 78

amounts of genetic material (in terms of

number of rules) between experts. Because

the rule sets of each of the pair of

experts chosen to undergo crossover were

unequal in length, one point for crossover

was chosen to be MOD the length of the

shorter expert and one MOD the longer

expert.. If both values turned out to be

less than the length of the shorter rule

set then a double crossover occurred: a

central list of rules was exchanged

between the two and both experts retained

their original size. If only one point

was less than the shorter rule set, then

only a single crossover occurred, with the

tail end of the shorter expert now grafted

on to what. was once the longer one, and

vice versa. Both offspring of each

inversion survived. Figures 2a and 2b

demonstrate the two types of crossovers.

R1A--R1B--R1C-l-R1D--R1E--R1F-l-R1G--R1H-

R1I--R1J--R1K

R2A--R2B--R2C-l-R2D--R2E--R2F-l-R2G

crossover of rule sets 1 & 2 yield

R1A--R1B--R1C-l-R2D--R2E--R2F-l-R1G--R1H-

R1I--R1J--R1K

R2A--R2B--R2C-l-R1D--R1E--R1F-l-R2G DOUBLE CROSSOVER -
CROSSOVER POINTS INDICATED BY l Figure 2a

R1A--R1B--R1C--R1D-l-R1E--R1F--R1G--R1H-

R1I--R1J--R1K

R2A--R2B--R2C--R2D-l-R2E--R2F--R2G

crossover of rule sets 1 & 2 yield

R1A--R1B--R1C--R1D-l-R2E--R2F--R2G

R2A--R2B--R2C--R2D-l-R1E--R1F--R1G--R1H-

R1I--R1J--R1K SINGLE CROSSOVER - CROSSOVER POINT INDICATED BY l Figure 2b THE ROLE OF THE CRITIC Experts, naturally, relate to some

field of expertise. The general set of

problems which these experts were to solve

involved a set of related problems, where a number of different tests could be undertaken in order to determine some information about the system. The goal of each expert was to generate the proper set of tests, in proper order, so that the correct solution for each presented problem would be obtained at a minimal cost in terms of testing expenses. The role of the critic was to apply each expert's rule set, in order, (and repeatedly if necessary), to each member of a library of actual cases, and to obtain thereby some figure of merit for each expert. If the conditions precedent in the rule were met, then the rule fired and the action called for was taken. If the rule action called for an observation about the case, information might be unmasked, and the expert's state of knowledge increased (but only if the information was available in the actual library case). Of course costs may have been incurred in making such observations. These costs could involve time, money, or injury to the subject being observed. Results might be inconclusive or even unavailable. Rule evaluation ceased when a decision rule fired. If the entire rule set was evaluated without a decision rule firing, then the set was reevaluated from its beginning (and at an additional cost), because the results of earlier fired rules were likely to have provided additional information that could result in different rules firing on subsequent passes. A maximum number of passes were allowed in order to ensure termination. A valuation of the expert's merit vis a vis that particular problem was made, and then the next problem in the set was presented to it, and so on. Obviously, the larger and more varied the set of problems, the more sophisticated and discriminating the expert. Creation of a library of problems under direct program control can be a tedious and error-prone process. In order to facilitate both entry of and changes to the problem set, GENES allowed creation of the problem set via the

normal Unix(TM) vi editor, and then fed the error-free result to the program. In our trial system we were interested in evaluating our experts on a dual basis. One consideration was how often they reached the correct result. The other was how little cost was expended in reaching this result. An apportionment between these two factors is clearly a value judgment that must be made for each particular expert system. Thus each expert accrued a certain cost in its decision making, that being the sum of dollar costs incurred in running tests, 79 Copyrighted Material degradation costs in the case of environmental misfortunes (e.g. a damaged drill in the instance of oil exploration) and penalties for bad evaluations. The merit of each expert was inversely related to the costs it incurred. TIME DEPENDENT EVALUATIONS Because the system we were studying was one where time was a factor, we introduced the notion of time dependency to our evaluations of the goodness of each expert. Many of the tests which could be performed by the experts would not yield immediate results. While such answers were pending other tests might or might not be undertaken, depending upon the rules (where actions might or might not be dependent upon previous test results). Time dependency required that the entire rule set for each expert be reevaluated repeatedly, because new tests might be ordered once earlier test results came in. In many existing systems such delay involves additional daily overhead costs, so these costs were factored into the testing costs for each expert where appropriate. A related notion was persistence, persistence being a quality associated with tests whose results do not change meaningfully over short intervals. In order to prevent the repeated ordering of such tests, they were each assigned a persistence value which was decremented with time. If a rule required that a test be done when that test showed a non-zero persistence value, the rule was ignored. It should also be noted that there was a possibility that test results might be equivocal. To simulate this possibility, the individual problems in the sets presented were issued certain flags representing both the current and possible results of tests and observations that the expert might make. Thus, for example, in a medical situation, the patient's sex might generally be immediately knowable, the results of a blood culture would be available in 24 hours, but meaningful results of a CAT scan might actually turn out to be unobtainable. Because there was no prohibition on the repetition of rules within an expert's repertory, however, the CAT scan could be repeated and might show results the second time. HANDLING CONVERGENCE Researchers in the genetic algorithm field have repeatedly noted the

difficulty of fine tuning the parameters in order to obtain results along the desired results. The algorithm shares this rather nasty fact with other types of computer programming: "what you ask for is what you get" (See, e.g., (8)). Due to the emphasis upon getting the correct result as compared to the cost of arriving at that result, the population of experts tended to converge rapidly to a rather homogeneous solution set, wherein a high percentage of the problem sets tested were solved properly, but at less than optimum cost. Rapid convergence is a common problem in these types of systems and has been solved in various ways (9). The solution used in GENES was to consider the system to have converged if the sum of all the expert scores remained within a narrowly bounded interval over several consecutive iterations. If the system was judged to have converged, then a percentage of the population was replaced by newly generated experts, thereby providing a major influx of genetic material into the system. PARAMETERS FOR GENETIC OPERATIONS Rates of mutation, inversion and crossover, as compared with just plain duplication, were tunable parameters, as was the population size. Typically, rates in the range recommended by Grefenstette (10), namely a 30 percent crossover rate, 3 percent mutation rate, S percent inversion rate, and a 40 percent generation gap, were found to give good results and were used. Variations in these rates affected the time it took to reach a good solution, but did not effect the goodness of that solution. Other typical variable values were a population of experts set at 50 and 50 maximum rules; this was with 13 possible actions that could be taken and 12 possible tests-tobe-done where the possible variations for the bound of each test-to-be-done were extremely large since the constants chosen were from a continuous rather than a discrete population (e.g. temperature > 102.45). This, along with the variable number of branches in each rule and the variable number of rules in each set, makes the size of the search space difficult to estimate. It certainly not small. Using these parameters, the choice or which experts were to undergo each procedure was made via the weighted roulette wheel. In all cases at least one copy of the expert with the best value was retained. RESULTS The GENES model, we tested at least on a small scale, converged in approximately 2000 iterations with a population size of 30, or in 8 v 80

iterations with a population size of 50,

to results which appear optimal in that

the correct decisions were made for all

nine problems presented and at a

reasonably low cost. This required no

excessive or duplicative testing. Due to

the very large possible variation of

rules, there did, however, seem to be some

strange rules that remained in the best

expert's rule sets. These rules tended

not to make a great deal of sense, but

they did not actually affect the system as

their very weirdness assured that they

either always or never fired. Thus the

rule interpreted as "if the patient's

temperature is less than 120.3 degrees

discover if abdominal pair, is present", is

practically equivalent to stating "always

check for abdominal pain". Additionally, it should be noted that

all rules following one that resulted in a

decision being made were never evaluated,

so they did not actually contribute to the

rule set, although they did provide

genetic material to succeeding generations

and might be activated during inversion. FURTHER DIRECTIONS This system is currently being

examined with promising results in two

areas of expertise. The first area that we are investigating is the minimization of hospital costs during patient diagnostic evaluations in a prospective reimbursement environment. The simple environment we are studying is a rule set appropriate to a patient admitted with suspected gall bladder disease. The expert in this system should be able to arrive at a correct diagnosis using minimal hospital resources in terms of tests and time. It should be noted that this system does not involve interpretation of tests, but only the suggests the order of testing, as a function of previous results. The second system under study involves server activation and deactivation in queueing problems. Costs here include both activation and use of the server, with the goal being minimization of costs, while providing adequate service to the queue under varying server loads. A back end interpreter to translate the rule sets into terms understandable by semi-expert users is also under consideration. ACKNOWLEDGMENT The authors wich to acknowledge the

assistance of Neal Coulter for his many

tracking vs. averaging, exploration vs. exploi-

tation, etc .?

(6) What combinations of operators yield

implicit parallelism? Traditional mathemat ics with its reliance

upon linearity, convergence, fixed points, and

the like, seems to offer few tools for studying

such questions. Yet, without a relevant

mathematical framework, there is less chance

of understanding ANNs than there would be of

understanding physical phenomena in the

absence of guidance from theoretical physics. A

mathematics that puts emphasis on combina-

torics and competition between parallel

processes is the key to understanding ANNs.

What seems startling when one uses differential

equations, where the emphasis is on con-

tinuity, is comonplace in a programming or

recursive format, where the emphasis is upon

combinatorics. (Consider, for example, the

chaotic regimes that are so unexpected in the

context of differential equations, but are an

everday occurrence, in the guise of biased

random number generators, in the program-

ming context.) Because classifier systems are formally defined and computer-oriented, with an emphasis on combination and competition, they offer a useful test-bed for both mathematical and simulation studies of ANNs. We already have some theorems that provide a deeper understanding of the behavior of classifier systems (see the Appendix), and simulations suggest a broader class of theorems that delineate the conditions under which internal models (q-morphisms) emerge in response to complex environments (Holland [1986b]). By putting classifier systems in a broader context, we can bring to bear relevant pieces of mathematics from other studies. For instance, in mathematical economics there are pieces of mathematics tha t deal with (l) hierarchical organization, (2) retained earnings (fitness) as a measure of past performance, (3) competition based on retained earnings, (4) dis t r ibut ion of e a r n i n g s on t h e basis of local interactions of consumers and suppliers, (5) taxation as a control on efficiency, and (6) division of effort between production and research (exploration versus exploration). Many of these fragments, mutat is mutandis, can be used to study the counterparts of these processes in other ANNs. As another example, in mathematical ecology there are pieces of mathematics dealing with (l) niche exploitation (models exploiting environmental

opportunities), (2) phylogenetic hierarchies, polymorphism and enforced diversity (competing subsystems), (3) func-tional convergence (similarities of subsystem organization enforced by environmental requirements on payoff a t t a inment) , (4) symbiosis, parasitism, and mimicry (couplings and interactions in a default hierarchy, such as an increased efficiency for extant generalists simply because related specialists exclude them from some regions in which they are inefficient), (5) food chains, predator-prey relations, and other energy transfers (appor-tionment of energy or payoff amongst component subsystems), (6) recombination of multifunctional co-adapted sets of genes (re-combination of building blocks), (7) assortative mating (biased or triggered recombination), (8) phenotypic markers affecting interspecies and intraspecies interactions (coupling), (9) "found-er" effects (generalists giving rise to specialists), and (10) other detailed commonalities such as tracking versus averaging over environmental changes (compensation for environmental variability), allelochemicals (cross-inhibition), linkage (association and encoding of features), and still others. Once again, though mathematical ecology is a young science, there is much in the mathematics that has been developed that is relevant to the study of other nonlinear systems far from equilibrium. The task of theory is to explain the pervasiveness of these features by elucidating the general mechanisms that assure their emergence and evolution. Properly applied to classifier systems, or to ANNs in general, such a theory militates against ad hoc solutions, assuring robustness and adaptability for the resulting organization. One of the best ways to insure that the mechanisms investigated are general is "to look over your shoulder" 83 frequently to see if the mechanisms apply to all ANNs. This view is sharpened if we pay close attention to features shared by all ANNs: (1) Hierarchical organization. All ANNs exhibit an hierarchical organization. In living systems proteins combine to form organelles, which combine to form cell types, and so on, through organs, organisms, species, and ultimately ecologies. Economies involve individuals, departments , divisions, companies, economic sectors, and so on, until one reaches national, regional, and world economies. A similar story can be told for each of the areas cited. These s t ructural similarities are more than super-ficial. A closer look shows tha t the hierarchies are constructed on a "building block" principle: Subsystems a t each level of the hierarchy are constructed by combination of small numbers of subsystems from the next lower level. Because even a small number of building

blocks can be combined in a great variety of ways there is a great space of subsystems to be tried, but the search is biased by the building blocks selected. At each level, there is a continued search for subsystems that will serve as suitable building blocks at the next level. (2) Competition. A still closer look shows that in all cases the search for building blocks is carried out by competition in a population of candidates. Moreover there is a strong relation between the level in the hierarchy and the amount of t ime it takes for competitions to be resolved -- ecologies work on a much longer t ime-scale than proteins, and world economies change much more slowly than the depar tments in a company. More carefully, if we associate random variables with subsystem ratings (say fitnesses), then the sampling ra te decreases as the level of the subsystem increases. As we will see, this has profound effects upon the w a y in which the system moves through the space of possibilities. (3) Game-like system/environment interaction. An ANN interacts with its environment in a game-like way: Sequences of action ("moves") occasionally produce payoff, special inputs tha t provide the system with the wherewithall for continued existence and adaptation. Usually payoff can be t reated as a simple quant i ty (energy in physics, fitness in genetics, money in economics, winnings in game theory, reward in psychology, error in control theory, e tc .) It is typical tha t payoff is sparsely distributed in the environment and that the adaptive system must compete for it with other systems in the environment. (4) Exploitation of regularities. The environment typically exhibits a range of regularities or niches tha t can be exploited by different action sequences or strategies. As a result the environment supports a variety of processes that interact in complex ways, much as in a multi-person game. Usually there is no super-process that can outcompete all others so an ecology results (domains in physics, interacting species in ecological genetics, companies in economics, cell assemblies in neurophysiological psychology, e t c .) . The very complexity of these interac-tions assures that even large systems over long time spans can have explored only a minuscule range of possibilities. Even for much-studied board games such as chess and go this is t rue , the not so simply defined "games" of ecological genetics, economic competition, immunogenesis, CNS activity, e tc . , are orders of magnitude more complex. As a consequence the systems are always far from any optimum or equilibrium situation. (5) Exploration vs. exploitation. There is a tradeoff between exploration and exploitation. In order to explore a new niche a system must use new and untried

action sequences tha t take it into new parts (state sets) of the enviroment. This can only occur at the cost of departing from action sequences that have well-established payoff rates. The ratio of exploration to exploitation in relation to the opportunities (niches) offered by the environment has much to do with the life history of a system. (6) Tracking vs. averaging. There is also a tradeoff between "tracking" and "averaging". Some parts of the environment, change so rapidly relative to a given subsystem's response ra te that the sub-systerm ean only t-eaet to the average effect, in other situations the subsystem can actually change fast enough to respond "move by move". Again the relative proportion of these two possibilities in the niches the subsystem inhabits has much to do with the subsystem's life history 84

(7) Nonlinearity. The value ("fitness") of a

given combination of building blocks often

cannot be predicted by a summing up of values

assigned to the component blocks. This

nonlinearity (commonly called epistasis in

genetics) leads to co-adapted sets of blocks

(alleles) that serve to bias sampling and add

additional layers to the hierarchy.

(8) Coupling. At all levels, the competitive

interactions give rise to counterparts of the

familiar interactions of population biology -

symbiosis , parasitism , competitive exclusion ,

and the like.

(9) Generalists and specialists. Subsystems can

often be usefully divided into generalists

(averaging over a wide variety of situations,

with a consequent high sampling rate and high

statistical confidence, at the cost of a relatively high error ra te in individual situations) and specialists (reacting to a restricted class of situations with a lowered error ra te , bought at the cost of a low sampling ra te) .

(10) Multifunctionality. Subsystems often exhibit multifunctionality in the sense that a given combination of building blocks can usefully exploit quite distinct niches (environmental regularities), typically with different efficiencies. Subsequent recombinations can produce specializations that emphasize one function, usually at the cost of the other. Extensive changes in behavior and efficiency, together with extensive adaptive radiation, can result from recombinations involving these multifunctional founders.

(11) Internal models. ANNs usually generate implicit internal models of their environments, models progressively revised and improved as the system accumulates experience. The systems learn. Consider the progressive improvements of the immune system when faced with antigens, and the fact that one can infer much about the system's environment

and history by looking at the antigen

population. This ability to infer something of a

system's environment and history from its

changing internal organization is the diagnostic

feature of an implicit internal model. The

models encountered are usually prescriptive -

they specify preferred responses to given

environmental states -- but, for more complex

systems (the CNS, for example), they m a y also be more broadly predictive, specifying the results of alternative courses of action. The relevant mathematical concept of a model of process-like transformations is tha t of a homomorphism. Real systems almost never admit of models meeting the requirements for a homomorphism ("commutativity of the diagram"), but there are weakenings, the so-called q-morphisms (quasi-homomorphsms). The origin of a hierarchy can be looked upon as a sequence of progressively refined q-morph-isms (specifically q-morphisms of Markov processes) based upon observation. Functional Extensions. The foregoing questions and commonalities, together with some of the problems already encountered in simulations, have already suggested extensions of the standard definitions (as in Holland [1980]) of classifiers systems. One important change involves the way bids are used in determining the winners of competititions for activation. The standard way of doing this is to calculate a bid = [bid ratio]*[strength]. Under this arrangement , the local fixed points of classifiers are such that a generalist and a specialist active in the same situations will come to bid the same amount (because the strength of the generalist increases to the point of compensating for its smaller bid ratio, see the Appendix). This goes against the dictum that specialists should be favored in a competition with generalists. To compensate for this an effective bid is calculated by reducing the bid in proportion to the general-ity of the classifier producing the bid. The effective bid is then used in determining the probability that the classifier generating it is one of the winners of the competition. If the classifier wins it must pay the bid, not the effective bid, to its suppliers under the bucket brigade. Thus, the local fixed points are

not changed, but specialists are favored in compe-tition with generalists. This change goes a long way toward reducing instabilities in emergent default hierarchies. (We are still exploring the effects in simulations and, at the level of theory, the resulting modifications in global fixed points). 85 A related change concerns the method of determining a classifier's probability of producing offspring, its fitness, under the genetic algorithm. The higher strength of a generalist at its local fixed point greatly favors it in the production of offspring, and simulations indicate that this overbiases the evolution of the system toward the offspring of generalists. The simplest way of compensating for this is to make the fitness proportional to [bid ratio]*[strength] ra ther than strength alone. In intuitive terms, this makes the fitness proportional to the classifier's potential for affecting the system (its bid can be thought of as a "phenotypic" effect), ra ther than its reserves (strength is a quant i ty determined by its "genotypic" fixed-point). We have yet to car ry out an organized set of simulations based on fitness so-determined.
Simulations have also revealed two other effects worth systematic investigation. The first of these is the "focussing" effect of the size of the message list (see R. Riolo's paper in this Proceedings). In effect, a small message list forces the system to concentrate on a few factors in the cur ren t situation. Clearly there is the possibility of making the size of the message list depend upon the "urgency" of the situation. For example, during "lookahead" the message list's size can be quite large to encourage an exploration of possibilities, while at "execution" t ime the size can be reduced to enforce a decision Clearly, the system can use classifiers to control the size of the message list. This makes the size dependent upon the system's "reading" of the current situation, and the "reading" is subject to long-term adaptive change under the genetic algorithm. A second simple effect is to revise the definition of the environment , or equivalently the definition of the system's speed, so that typical stimuli persist for several time-steps. (This corresponds to the fact that the CNS operates rapidly relative to typical changes in its environment -- usually, milliseconds vs. tenths of a second.) The resulting "persistence" and "overlap" of input messages makes it much easier for the classifier system to develop causal models and associative links (see below). As yet, to my knowledge, no simulations have been built along these lines. At a much more general (and speculative) level, use of triggered genetic operators pro-vides a major extension of genetic algorithms.

Triggering amounts to invoking genetic operators with selected arguments , when certain pre-defined conditions are satisfied. As an example of a triggering condition consider the following: "Only general classifiers that produce weak bids are activated by the current input message." When this condition occurs it is a sign that the system has little specific information for dealing with the current environmental situation. Let this condition trigger a cross between the input message and the condition parts of some of the active general rules. The result will be plaus-ible new rules with more specific conditions. This amounts to a bot tom-up procedure for producing candidate rules that will automati-cally be tested for usefulness when similar situations recur. As another example of a triggering condition consider: "Rule C has just made a large profit under the bucket brigade." Satisfaction of this condition signals a propitious time to couple the profitable classifier to its stage -setting precursor An appropriate cross between the message part of a rule C Q active on the immediately preceding t ime-s tep -- the precursor -- and the condition part of the profit-making successor can produce a new pair of coupled rules (The trigger is not acti-vated if C Q is already coupled to C). The coupled, offspring pair models the state transi-tion mediated by the original pair of (uncoupled) rules. Such coupled rules can serve as the building blocks for models of the environment. Because the couplings serve as "bridges" for the beset brigade, these building blocks will be assig credit in accord with the efficacy of the mc constructed from them Interestingly enoug: nere seems to be a rather small number of robust triggering conditions (see Holland et al. [1986]), but each of them 86

would appear to add substantially to the

responsiveness of the classifier system. Tags are particularly affected by triggering

conditions tha t provide new couplings. Tags

serve as the glue of larger systems, providing

both associative and temporal (model-building)

pointers. Under certain kinds of triggered

coupling the message sent by the precursor in

the coupled pair can have a "hash-coded"

section (say a prefix or suffix). The purpose of this hash-coded tag is to prevent accidental eavesdropping by other classifiers -- a sufficient number of randomly generated bits in the tag will prevent accidental matches with other conditions (unless the tag region in the condition part of the potential eavesdropper consists mostly * 's) . If the coupled pair proves useful to the system then it will have further offspring under the genetic algorithm, and these offspring often will be coupled to other rules in the system. Typically, the tag will be passed on to the offspring, serving as a common element in all the couplings. The tag will only persist if the resulting cluster of rules proves to be a useful "subroutine". In this case, the "subroutine" can be "called" by messages tha t incorporate the tag, because the conditions of the rules in the cluster are satisfied by such messages. In short, the tag that was initially determined at random now "names" the developing subroutine. It even has a meaning in te rms of the actions it calls forth. Moreover, the tag is subject to the same kinds

of recombination as other parts of the rules (it is, after all, a schema). As such it can serve as a building block for other tags. It is as if the system were inventing symbols for its internal use. Clearly, any simulation tha t provides for a test of these ideas will be an order of magnitude more sophisticated than anything we have tried to date. Runs involving hundreds of thousands of t ime-steps and thousands of classifiers will probably be required to test these ideas. Support is another technique tha t adds considerably to the system's flexibility. Basically, support is a technique tha t enables the classifier system to integrate many pieces of partial information (such as several views of a partially obscured object) to arrive a t strong conclusions. Support is a quanti ty tha t travels with messages, ra ther than being a counterflow as in the case of bids. When a classifier is satisfied by several messages from the message list, each such message adds its support into tha t classifier's support counter. Unlike a classifier's strength, the support accrued by a classifier lasts for only the time -step in which it is accumulated. That is, the support counter is reset at the end of each time-step (other techniques are possible, such as a long or short half-life). Support is used to modify the size of the classifier's bid on tha t t ime-step, large support increases the bid, small support decreases it. If the classifier wins the bidding competition, the message it posts carries a support proportional to the size of its bid. The propagation of support over sets of coupled classifiers acts somewhat like spreading activation (see Anderson [1983]), but it is much more directed. It can bring associ-ations (coupled rules) into play while serving its pr imary mission of integrating partial infor-mation (messages from several weakly-bidding, general rules tha t satisfy the same classifier). In addition to these broadly conceived extensions, there are

more special extensions that may have global consequences, partic-ularly in respect to increased responsiveness and robustness. One of these concerns a simple redefinition of classifiers. The standard definition of a 2-condition classifier requires tha t each condition be satisfied by some message on the message list, in effect an AND, requiring a message of type X and a message of type Y. It is a simple thing to replace the implicit AND with other string operators, e.g. a b i t -by-bi t AND or a binary sum of the satisfying messages, which is then passed through as the outgoing message. This extension has been implemented, but has not been systematically tested. Other simple extensions impact the functioning of the genetic algorithm. It is easy to introduce, in the string defining a classifier, punctuation marks tha t bias the probability of 87 crossover (say crossover is twice as likely to take place adjacent to a punctuation mark) These punctuation marks are not interpreted in executing the classifier, but they bias the form of its offspring under the genetic algorithm. Punctuation marks can be treated as alleles under the genetic algorithm, subject to muta-tion, crossover, e t c . , jus t as the other (func-tion-defining) alleles. This ensures that the placement of punctuation marks is adaptively determined. Similarly one can introduce mat-ing tags t ha t restrict crossover to classifiers with similar tags, again the tags, as part of the classifier, can be made subject to modi-fication and selection by the genetic algorithm. Finally, there are two broad ranges of investigation, far beyond anything we yet understand either theoretically or empirically, tha t offer intriguing possibilities for the future. One of these s tems from the fact that classifier systems are general -purpose. They can be programmed initially to implement whatever expert knowledge is available to the designer, learning then allows the system to expand, correct errors, and transfer information from one domain to another . It is important to provide ways of instructing such systems so tha t they can generate rules -- tentative hypotheses -- on the basis of advice. It is also important tha t we understand how lookahead and vir tual explorations can be incorporated without disturbing other activities of the sys-tem. Little has been done in either direction. The other realm of investigation concerns fully-directed rule generation. In a precursor of classifier systems, the broadcast language (Holland [1975]), provision was made for the generation of rules by other rules. With minor changes to the definition of classifier systems, this possibility can be reintroduced. (Both messages and rules a re strings. By enlarging the message alphabet, lengthening the message string, and

introducing a special symbol tha t indicates whether a string is to be interpreted as a rule or a message, the task can be accomplished.) With this provision the system can invent its own candidate operators and rules of inference. Survival of these m e t a (operator -like) rules should then be made to depend on the net usefulness of the rules they generate (much as a schema takes it value from the average value of its carr iers) . It is probably a ma t t e r of a decade or two before we can do anything useful in this area. Mathematical Extensions. There are at least two broader mathematical tasks that should be undertaken. One is an a t tempt to produce a general characterization of systems that exhibit implicit parallelism. Up to now all such a t tempts have led to sets of algorithms that are easily recast as genetic algorithms -- in effect, we still only know of one example of an algorithm that exhibits implicit parallelism. The second task involves developing a mathematical formulation of the process whereby a system develops a useful internal model of an environment exhibiting perpetual novelty. In our (preliminary) experiments to date, these models typically exhibit a (tangled) hierarchical s t ructure with associative coup-lings. As mentioned earlier, such structures can be characterized mathematically as quasi-homomorphisms (see Holland et al. [1986]). The perpetual novelty of the environment can be characterized by a Markov process in which each state has a recurrence time tha t is large relative to any feasible observation t ime. Con-siderable progress can be made along these lines (see Holland [1986b]), but much remains to be done. In particular, we a need to construct an interlocking set of theorems based on: (1) a more global set of fixed point theorems that relates the strengths of classifiers under the bucket brigade to observed payoff statistics; (2) a set of theorems that relates building blocks exploited by the "slow" dynamics of the genetic algorithm to the sampling rates for rules at different levels of the emerging default hierarchy (more general rules are tested more o f t r n) , and (3) a set of theorems (based on the previous two sets) that detail the way in which various kinds of environmental regularities are exploited by the genetic algorithm acting in terms of the strengths assigned by the bucket brigade 88

Appendix.

A simplified version of the fundamental

theorem for genetic algorithms can be stated as

follows (for an explanation of terms, see

Holland [1975] or Holland [1986a]).

Theorem (Implicit parallelism). Given a

fitness function u: (0,1) k Reals + , a population

B(t) of M strings drawn from the set (0,1) k ,

and any schema s ε(0,l,#) k defining a hyper-

plane in {0,l} k , M s (t+1) > u ' S (t) ε s M S (t) ,

where M s (t+l) is the expected number of

instances of s in B(t+l), u' s (t)= Σ bεsANDbεB(t) U(b)/m
s (t),

is the average observed fitness of the instances

of schema s in B(t), and ε s = (1-(k s -1)P cross)/(k-1)),

is a "copying error" induced by crossover,

where P cross is a constant of the genetic algor

ithm (often p cross = 1) giving the proportion of

strings undergoing crossover in a given genera

tion, and k 5 l is the number of crossover

points between the outermost defining symbols

of se(0,1,#) k . Under interpretation, the implicit
parallel

ism theorem says that the sampling rate for

every schema with instances in the population

is expected to increase or decrease at a rate

specified by its observed average fitness, with

an error proportional to its defining length.

Theorem (Speedup). The number of schemas

processed with an error < ε under a genetic

algorithm considerably exceeds M 3 for a

population of size M = 2 1/2k ', where ε = k'/k.

Theorem (Bucket brigade local fixed-point) .

If, under the bucket brigade algorithm, I c is

the long-term average income (after taxes) of

a classifier C, and r c is its bid-ratio, then its

strength S c will approach l c /r c

Theorem (q-morphism parsimony ; for defi

nitions, see Holland et al. [1986]). A

q-morphism of n levels, in which each succes

Greeqy,Genetics Although they did not call it such, Grefenstette

et al (1985) developed a greedy crossover for the

traveling salesman problem:

1. For each pair of parents, pick a random city for the start.

2. Compare the two edges leaving the city (as represented in the two parents) and choose the shorter edge.

3. If the shorter parental edge would introduce a cycle into the partial tour, then extend the tour by a randan edge.

4. Continue to extend the partial tour using steps two and tnree until the circuit in completed. Grefenstette et al. applied their heuristic to

a 50 city and a 100 city problem. Unfortunately,

the results of Grefenstette et al. were not

directly comparable to those of Goidberq and

Lingle. It might be expected that an analysis of

schemata survivability (Goidberq, 1906) or a hiii-
climbing variant (Ackiey, 1967) might be inves
tigated for the greedy crossover. A little
reflection suqqests that the former is signifi
cantlv more difficult tnan for PMX or traditional
crossover operators. Alternating hili-climbing
with genetic algorithms has produced successful
results when interim populations suggested by the
genetic algorithm provide good initial points for
hill-climbing. However, the greedy algorithm (as
implemented here) is not a "hill-climber", nor does
it benefit from different starting points. On the
other hand, Markov chain analysis (Goldberg and
Segrest 1987) might Drovide insight into greedy
genetics; such studies are expected to be pursued
in later papers.

Implementation The comparative analysis of conventional and
greedy genetics was performed on three classes of
problems: set covering (SCP), job shab scheduling
(JSS), and traveling salesman (TSP). The genetic
algorithm used was the Genesis system
(Grefenstette 1986) modified to generate a single
offspring for each crossover operation. This
offspring replaced the most similar of its parents.
(One offspring per crossover was mandated by the

greedy genetics and was used in all crossover

methods to insure comparability. Replacement of

the most similar parent provided a simple yet

efficient means of avoiding premature convergence-

- a pervasive problem with greedy genetics.)

Additional modifications were made to accommodate

the integer coding underlying JSS and TSP. (No

such modification was required for SCP.) PMX and

qreerdy crossover operations were added to Genesis.

A fixed population size of 75 was used for SCp and

candidate solutions were reoresented as binary strings of length 25 (the SCP matrices were of dimension 50 x 25). A fixed population size of 50 was used for TSP and JSS for the majority of the experiments and the candidate solutions were represented as permutations of 15 objects (JSS and TSP problems involved 15 jobs and 15 cities, respectively). (Selected experiments were run with population sizes 100 and 250. The results differed little from those presented here.) Genesis parameters not explicitly discussed in the previous paragraphs were left at their default settings. For each of the problem classes, a sample of problems were generated and the problems were attacked with both conventional and greedy versions of the genetic algorithm. Comparative performance of the best solution and trial at which it was achieved; as well as on-line, off-line, and average performance were all investigated. Golden and Stewart (1985) suggest a number of statistical tests to evaluate the relative performance of competing algorithms. They suggest the Wilcoxon signed rank test, the Friedman test, and an expected utility approach. Such tests are useful as confirmatory statistical analysis. However, the results of this paper are still considered prelimi nary and it is felt that little would be gained by additional statistical formalism. Set-Covering The set-covering problem (SCP) is NP-comolete (Garey and Johnson 1979) and is often encountered in applications such as resource allocation (Revelle et al 1970) and scheduling (Marsten and Shepardson, 1981). Known solution techniques for SCP

include such methods as integer programming; heuristic branch and bound; and most recently, Lagrangian relaxation with subgradient variations -- see Balas and Ho (1980) for a review of set covering solution techniques. The usual representation of a set coverinq problem is as a zero/one matrix with a cost associated with each column. A set of columns is defined to cover the matrix if for each row of the matrix at least one of the columns of the set contains a '1' entry. The SCP objective is to find a minimal cost cover of the matrix as follows: Let A be an mxh binary matrix and w i , i=1, ...,n non-negative costs. n Minimize (over δ) Σ w i δ i i=l Subject to A δ > 1 where a. 0 = (δ 1 , δ n) t b. 1 is an m-dimensionai column vector of ones, and c. each δ i is binary. 91 The genetic algorithm representation of SCP is straightforward: a candidate SCP solution is expressed as a binary string where a ' 1' in position i indicates that the ith column of the matrix is included in the solution. The reward is a large number M minus the sum of the cost of the columns used and a penalty function Pn for failure to cover: n R = M Σ w i δ i P n i=l where δ i and n are binary with n = 0 if the solution is a cover and n = 1 otherwise, and P is an appropriate scaling for the penalty. (The choice of the penalty scaling factor P is known to affect the performance of the genetic algorithm SCP performance, but will not be investigated in this paper.) Each of single crossover, double crossover, and greedy crossover was investigated. The latter was defined as follows: For every pair of parent genes, 1. Initialize the set S to be empty and the matrix A to be the original set covering matrix with columns {c i } and associated costs {w i }. 2. For the unused columns and uncovered rows calculate the cost-ratio (cost/number-of-rows -covered = W i /number-of-rows-covered). 3. Append to S the column (say column c) with the least cost that is included in one of the parents. 4. Strike column c and the rows covered by c from A and let this new matrix be A'. 5. If S is a cover or if no other columns are represented in the parents, stop. Otherwise, set A to A' and go to step 2. The greedy SCP crossover is illustrated in Figure 1. parent 1: 1 0 0 0 0 w 1 =5 w 2 =2 w 2 =5 w 4 =4 w 5 =2 parent 2: 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1 initial selection 0 0 0 1 0 for child: column 1 0 0 1 0 2 with cost-ratio 1 1 second-stage cost-ratio second selection: column 1 column 3 column 4 column 5 column 1 5 °° 2 °° 1 1 Note: Column 4 was not selected because it was not represented in either parent. Figure 1. Greedy crossover for SCP. The experimental design for the set-covering problem involved matrix density as an additional parameter. (Matrix density for a zero-one matrix as the proportion of ones in the

matrix.) Five test problems were generated for each of the (expected) densities ranging from 10% to 90%. The results are displayed below: Table 1 presents the relative success of each of the methods on each of the classes of the problems and Table 2 presents a problem by problem breakdown of the performance. Generation average performance and best of genera tion performance was displayed for representative matrices of 30% and 70% density in Figures 2-5. CROSSOVER METHOD PROBLEM DENSITY one-point wins ties two-poant wins ties 10 1 4 20 1 _ 4 1 30 2 3 2 40 1 1 3 1 50 2 3 2 greedy wins ties 60 1 4 1 70 1 2 3 2 80 2 2 2 3 90 1 1 3 2 Totals 1 6 1 10 29 14 Table 1. Relative SCP performance by problem density and crossover method. The major conclusion that can be drawn from the SCP results is that the greedy genetics for the set-covering problem not only yield a better solution than either the pure greedy or conven tional genetics, but that convergence to that solution is much more raoid than with conventional genetics. Job Shop Scheduling The JSS problem (just like the TSP) problem is a pure ordering problem. Since it is not guaran teed to result in an order, conventional crossover is not applicable to ordering problems unless the problems are formulated as a penalty function problem. Otherwise, mechanisms such as PMX must be used and the underlying genetic building blocks are the o-schemata. Bethke (1980) analyzed GA-hard functions using Walsh transforms, group characters of the group $\Pi Z 2$, the n-fold Cartesian product of the group Z 2 with component-wise addition (Dym and McKean 1972). A natural extension of Bethke's work to the analysis of "GA-order-hard" functions might be based on the group representations of the symmetric group (Boerner 1963). Three types of crossover operators were investigated for job shop scheduling: PMX, a weak greedy crossover, and a powerful greedy crossover. The job shop scheduling problem investigated was the simplest scheduling problem: a static queue of jobs with specified due dates and run times with no precedence constraints, a single server and minimal 92 1 Pt. Cross 2 Pt. Cross Greedy Cross Pure @ Best Trial Best Trial Best Trial Greedy Ontimal 1: 61.29 412 62.03 560 60.11 80 60.11 ? 2: 45.72 593 47.79 593 48.2 56 48.76 ? 10% 3: 48.31 613 43.16 630 43.28 258 44.42 ? 4: 49.72 702 51.58 663 49.11 90 49.82 ? 5: 55.79 547 50.85 389 48.38 77 48.38 7 1: 19.36 682 18.78 831 18.78 61 22.33 ? 2: 34.54 550 33.6 496 29.34 669 31.27 7 20% 3: 33.66 713 38.37 524 27.49 114 30.25 ? 4: 26.42 534 27.02 783 22.22 646 24.17 7 5: 24.09 614 16.23 641 13.41 75 16.03 ? 1: 16.30 786 11.52 787 11.52 77 11.52 11.52 2: 20.13 918 16.70 733 16.06 93

17.19 14.80 30% 3: 8.46 611 11.85 913 8.20 85 8.20
8.20 4: 17.77 763 18.10 431 18.10 108 20.06 ? 5:
9.99 688 8.83 915 7.55 96 7.57 7.55 1: 10.73 937
11.58 788 11.53 81 11.53 10.70 2: 12.21 791 12.97 745
9.78 108 11.62 9.56 40% 3: 11.38 949 11.70 1006 7.50
95 7.50 7.50 4: 7.99 523 6.45 633 6.45 96 6.82
6.42 5: 17.95 631 15.26 612 13.31 104 16.00 12.64
1: 3.62 703 5.45 623 2.86 78 3.62 2.86 2: 4.38 378
5.40 434 4.38 79 4.38 4.38 50% 3: 8.07 855 11.16 684
7.67 81 7.57 7.57 4: 6.10 992 8.91 718 6.10 220
7.90 6.10 5: 6.39 929 6.39 644 6.27 90 6.39 6.19 1:
4.75 893 4.24 913 3.12 90 3.28 3.12 2: 6.49 687 4.66
581 4.66 1014 5.23 4.66 60% 3: 6.14 723 6.14 962 5.86
77 5.86 5.31 4: 5.15 799 3.64 795 3.63 82 3.63 3.63
5: 6.93 493 4.05 779 3.77 76 5.12 3.77 1: 4.78 645 4.49
607 4.49 107 5.76 3.96 2: 3.42 663 2.15 795 1.76 94
2.04 1.76 70% 3: 2.31 547 2.31 729 2.31 76 2.31 2.31
4: 5.21 727 3.00 610 2.64 97 2.78 2.64 5: 0.96 473
0.96 540 0.61 77 0.76 0.61 1: 0.92 728 2.09 802 0.92
78 0.92 0.92 2: 2.03 426 2.03 922 2.03 92 2.69 2.03
80% 3: 3.89 442 3.41 548 2.25 85 2.92 2.25 4: 4.85 545
5.46 922 4.13 80 4.18 3.70 5: 3.58 819 2.59 686 2.59
80 3.58 2.59 1: 3.08 887 0.62 551 0.62 76 0.78 0.62
2: 0.66 979 1.19 787 0.66 86 0.68 0.66 90% 3: 0.73
851 0.54 815 0.51 79 0.72 ? 4: 3.82 667 3.81 532
3.65 80 5.20 3.65 5: 1.33 738 2.34 825 1.04 76 1.04
1.04 @ Greedy heuristic applied recursively until a cover
is generated. Table 2. Problem by problem SCP performance.
93 (signed) lateness as the criterion. (The more
complicated job shop scheduling problem considered by
Davis (1985) was not investigated.) A well known heuristic
provides the optimal job schedule for this problem: "Order
the queue according to increasing run times." The two
versions of the greedy crossovers were based on weak and
powerful heuristics, respectively. The powerful heuristic
is the optimal heuristic previously discussed; the weak
heuristic states "order the gueue according to increasing
difference between due date and run time"; that is, a job
with an early due date and a long run time would be
scheduled before a job with a late due date and a short
run time. The greedy crossovers are implemented as
follows: 1. For each parent pair, start at the first job
(i=l). 2. Compare the two jobs at the ith position of the
two parents and place the better (according to the
heuristic being used) of the two in the child's ith
position. If one of the jobs has already been placed in
the child, then automat-ically pick the other. If both of
the jobs have already been placed, then pick a job
randomly from the yet unplaced jobs. 3. Repeat step 2,
incrementing i, until ail positions in the child string

are defined. Results from the job shop scheduling experi-ments are presented in Tabie 3 and generation average and best of generation performance for each of the crossover methods for a representative JSS problem are graphically presented in Figures 6 and 7. It is obvious that the strong greedy crossover dominates PMX, which in turn dominates the weak greedy crossover. Repeated application of the powerful heuristic (pure strong greegy) would yield the optimal schedule with an evaluation of 0.00. 94 Copyrighted Material Figure 4. Generation average performance. Figure 3. Best of generation. Figure 2: Generation average performance Figure 5. Sest of generation PMX Weak Greedy Strong Greedy Best Trial Best Trial Best Trial 1 40.84 825 124.41 314 0.00 514 2 33.78 913 51.42 951 8.82 321 3 29.74 938 57.71 948 2.27 397 4 59.60 873 86.93 984 6.99 370 5 27.22 861 55.94 972 2.75 576 6 34.94 967 68.76 849 3.34 462 7 20.90 689 71.71 926 2.20 938 8 14.68 890 112.84 887 0.00 545 9 53.60 695 101.93 923 2.20 881 10 13.52 994 203.40 169 0.00 432 11 28.99 946 74.98 711 0.00 416 12 48.14 835 115.14 422 0.00 1005 13 18.02 941 63.18 955 1.60 481 14 17.57 701 80.20 967 1.24 720 15 15.07 976 150.09 397 10.85 424 16 7.93 946 136.03 492 2.23 599 17 50.18 786 208.27 43 3.24 959 18 31.35 1007 122.13 29 4.71 828 19 25.97 904 184.46 281 2.49 344 20 46.00 793 183.39 681 3.78 937 Table 3. Problem by problem JSS performance Traveling Salesman Problem (TSP) The traveling salesman problem (TSP) is another of the very difficult NP complete combinatorial ordering problems with a long history of interest. Few (1955) discovered a heuristic solution for the Euclidean problem with a guaranteed worst case performance of √2n + 1.75. Christofides (1976) discovered an elegant heuristic with a worst case error of 50% of optimal tour length. Karp (1977) showed that partitioning into subsets and con-catenating optimal tours of each subset yields tours with expected percentage error approaching zero. Crowder and Padberg (1980) solved a 318 city tour to optimality, and Golden and Stewart (1985) reported excellent results for their CCA0 heuris-tic. Of the many theorems and results relating to the TSP, two are of particular relevance to this paper. Theorem 1 (Rosenkrantz, Stearns, and Lewis 1977). For every r > 1, there exist n-city instances of the TSP for arbitrarily large n oPeying the triangle inequality such that NN(I) > r OPT(I), where NN(I) is the nearest neighbor optimal tour and 0PT(I) is the overall optimal tour. 95 Figure 7'. Best of generation. Figure 6: Generation average. pmx pmx greedy problem nearest unanchored anchored pmx genetic greedy # neighbor and seeded and seeded anchored seeded genetic 1 19484 19374 b

18243 b 23269 19211 b 18243 ab 2 16480 a 16480 a
16480 a 18096 16480 a 16480 a 3 17673 a 17673 a 17673
22588 17673 a 17815 4 15701 15626 b 14837 b 18495
14331 b 14256 ab 5 16219 16210 ab 16219 18569 16219
16219 6 13613 a 13613 a 13613 a 13613 13613 a 13916
7 17400 a 17400 a 17400 a 24578 17400 a 17745 8
15067 a 15067 a 15067 a 16714 15067 a 15067 a 9 19755
19755 10577 b 25129 19555 ab 19661 ab 10 21299 21299
20739 b 23392 20778 19821 ab 11 18034 17711 b 17740 b
22150 17364 17191 ab 12 18587 a 18587 a 18587 a 21998
18587 a 19708 13 17078 17078 17078 21063 17078 15759
ab 14 19535 19472 b 19205 b 23845 18581 ab 18581 ab
15 16052 a 16052 a 16052 a 20856 16042 a 16052 a 16
21934 20228 b 19897 b 21442 b 19253 b 18971 ab 17
16354 16354 16229 ab 20352 16229 ab 16354 18 21977
21977 21977 26089 21807a b 22668 19 17049 17006 ab
17015 b 22651 17049 18519 20 17869 a 17869 a 17869 a
25119 17869 a 17869 a Table 4. Problem by Droblem TSP
performance a best performance Theorem 2 (Papadimitriou
and Steiglitz 1977). If A is a local search algorithm
whose neighborhood search time is bounded by a polynomial
of the problem representation, then if P ≠. NP, A cannot
be guaranteed to find a tour whose length is bounded by a
constant multiple of the optimal tour even with an
exponential number of iterations. Theorem 1 is relevant to
this paper because the standard against which the genetic
algorithms are compared is the pure greedy algorithm,
called the nearest neighbor solution in the operation
research literature. The implication is that this standard
itself could be poor. Theorem 2 suggests that if the
greedy genetic algorithm is a local search algorithm as is
believed, then examples of tours can be found for which
the algorithm performs arbitrarily poprly. Presumably, the
same conclusions would hold for PMX. The work presented
here builds on that of Brockus (1983), Goldberg and Lingle
(1985), and Grefenstette et al (1985). The results of
those three studies were not directly comparable and it
is interesting to ask how they compare. Moreover, none of
the previous studies investigated the effect of seeding
the initial populations, of anchoring the tours, or of
stochastic variation due to different randomization. Two
tyoes of crossover operators were inves tigated for the
TSP: PMX and a modification of the greedy crossover of
Grefenstette et al (1985). For the anchored case all
members of the populatior were normalized to begin with
the same starting city (city '1'). For the unanchorea case
this was not done and the starting cities were randomly
chosen. For any two parents, the modified Grefenstette
crossover (greedy crossover) produces a child according to
the following specifications: 1. For a pair of parents,

start at the first position (always the same city). 2. Choose the shortest edge leading from the current city (that is represented in the parents). If this edge leads to a cycle, choose the other edge. If this leads to a cycle, choose a random citv that continues the tour. 3. If the tour is complete, stop; else go to 2. 96 b performance surpasses nearest neicrhbor The modified Grefenstette crossover will be

called the greedy crossover and is always anchored

in the results presented here. Both greedy

crossover and PMX crossover were tested with seeded

and unseeded initial populations. The seeded

initial population included the fifteen optimal

qreedy algorithm generated tours (possibly in-

cluding duplicates) beginning at each of the

fifteen different cities. This subpopulation of

fifteen was randomly extended to an initial

population of fifty. The nearest neighbor tour was

the best of the fifteen greedy algorithm generated

tours. Results of the performance of the qreegy

and PMX crossovers are qiven in Table 4. Dis

couraqing is the poor performance of the PMX with

the unseeded population; only for one problem did

it perform better than the nearest neighbor

algorithm. Virtually no performance differential

among the remaining variations of the genetic

algorithm was observed; each performed somewhat

better than the nearest neighbor. Perhaps surpris

ingly, the unseeded qreedy genetic performed nearly

as well as its ,seeded counterpart. Relative performance between anchored and

unanchored PMX and seeded and unseeded greedy

crossover is presented in Table 5, where the 2 in

the upper left hand corner indicates that the

unanchored PMX performed better than the anchored

PMX on two problems. The remaining entries are

interpreted similarly. Apparently, no important

differences are indicated. Figures 8 and 9

represent generation average performance and best

of generation results for the fourth problem at 70%

density. Bethke (1980) has shown that for certain

problems, genetic algorithms are unstable, that is,

the solution determined seems to depend on genetic

drift and on the randomization of the initial

population and genetic mechanism. Tables 6 and 7

display the results of different randomization of

initial populations and genetic mechanisms for the

first problem at 10% density. piscouragingly, ap-

proximately 8% variation in performance can be

noted for the greedy genetic as either initial

population or genetic mechanism is randomized

differently. The counterpart variations for the

PMX are 13% and 17%, respectively. The implication

seems to be that the TS? is a difficult problem for

either the PMX or qreedy gerietics. anx unanchored pmx anchored and seeded and seeded 2 6 greedy qenet ic greedy qenet ic seeded unseeded 8 6 Table 5. Seeding and anchoring as factors in performance differential population seed # pmx greedy genetic 1 2 3 4 5 22671 22445 21247 22456 24075 19636 18243 18733 18771 18858 Table 6. Random initial population seeds as a factor in performance differential crossover greedy seed # pmx genetic 1 24029 19537 2 23244 20721 3 22167 20713 4 24074 20275 5 22427 19072 6 21460 20078 7 22898 19600 8 21628 19348 9 22671 19636 10 25125 20415 Table 7. Random seeds for GA mechanism as a factor in performance differential 97 Figure 9. Best of generation for unseeded initial populations. figure 8. Generation average performance for unseeded Initial Population Conclusions and Future Research Although little effort was made to oDtimize the algorithms investigated in this paper, the evidence suggests that greedy genetics can successfully make use of problem specific information whenever the underlying greedy algorithm is powerful. (The underlying greedy could be defined to be "powerful" whenever a single problem application of it results in a "reasonable" solution.) On the other hand, if the underlying greedy is misdirected, greedy genetics are less successful than traditional genetics. However, regardless of the power of the underlying greedy algorithm, greedy genetics often converge more rapidly that their conventional counterparts. It appears that greedy genetics have their place in optimization and it would be interesting to extend this work to more realistic problems (such as job shop scheduling problems with prece-dence constraints and multiple servers) and to compare performance with more traditional optimiza-tion techniques. The variance in performance due to randomization. Markov chain results, extensions to Bethke's work, and oroper penalty function formulations all deserve additional analysis. RERERENCES Ackley, D. H. (1987). A Connectionist Machine for Genetic Hillclimbing. Kluwer Academic Publishers, Boston, MA. Balas, E. and A. Ho (1980). "Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization: A Computational Study," Mathematical Programming 12, 37-60. Bethke, A. D. (1980). Genetic Algorithms as Function Optimizers, Ph.D. Thesis, University of Michigan, Ann Arbor. Boerner, H. (1963). Representations of Groups with Special Considerations for the Needs of Modern Physics, North Holland Publishers, Amsterdam. Brockus, C. G. (1983). "Shortest Path Optimization Using a Genetics Search Technigue", Proceedings of the Fourteenth Annual Modeling and Simulation Conference, Pittsburgh, PA. 241-5.

Christofides, N. (1976). Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem, Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA. Crowder, H. and M. W. Padberg. (1980). "Solving Large-Scale Symmetric Traveling Salesman Problems to Optimality, Management Science, 26, 495-509. Davis, L., (1985). "Job Shop Scheduling with Genetic Algorithms," Proceedings of an Interna-tional Conference on Genetic Algorithms and tneir Applications, Carnegie-Mellon University, Pittsburah. DeJong, K. A. (1975). An Analysis of the Beriavior of a Class of Genetic Anaptive Systems," Ph.D. Thesis, University of Michigan, Ann Arbor. Dym, H. and H. P. McKean (1972). Fourier Series and Integrals. Academic Press, New York. Few, L. (1955). "The Shortest Path and the Shortest Road through n Points", Mathematika. 2, 141-144. Garey, M. R., and D. S. Johnson, (1979). Computers and Intractability: A Guide to the Theory of NP- Completeness. Freeman, San Francisco. Grefenstette, J. J., Gopal, B. J'. Rosmaita, and D. V. Gudht, (July 1985). "Genetic Algorithms for the Traveling Salesman Problem," Proceedings of an International Conference on Genetic; Algorithms and Their Applications, Carnegie-Mellon Univer-sity, Pittsburgh. Grefenstette, J. J., (April 1986). A User's Guide to GENESIS, Technical Report CS-84-11, Computer Science Department, Vanderbilt University, Nashville. Goldberg, D. E. (1986). Simple Genetic Algorithms and the Minimal Deceptive Problem, The Clearing-house for Genetic Algorithms, TOGA Report No. 86003, University of Alabama, Tuscaloosa. Goldberg, D. E., and R. Lingle, (July 1985). "Alleles, Loci, and the Traveling Salesman Problem," Proceedings of a International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh. Goldberg, D. E., and P. D. Segrest, (to appear) "Finite Markoy Chain Analysis of Genetic Algorithms", University of Alabama, Tuscaloosa. Golden, B. L., and W. R. Stewart. (1985). Empirical Analysis of Heuristics in The Traveling Salesman Problem, Lawler et al (eds), John Wiley and Sons, 207-250. Holland, J. H. (1975). Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor. Karp, R. M. (1977). "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problems in the Plane, Math Operations Research 2, 209-224. Lawler, E. L. (1976). Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston. Lawier, E. H., J. K. Lenstra, A. H. G. Rinoovkan, and D. B. Shmovs, (1985). The Traveling Salesman Problem, John Wiley and Sons. 98

Liepins, G. E., and M. R. Hilliard. (1986)

"Representational Issues in Machine Learning", Proceedings of the International Symposium on Methodologies for Intelligent Systems Colloguia Program. Knoxville, TN. ORNL-6362.

Marsten, R. F., and F. Shepardson, 1981. "Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications," Networks, Vol. 11, No. 2, pp. 165-177.

Nilsson, N. J. (1980). Principles of Artificial Intelligence, Tioga Publishing Company, Palo Alto, CA.

Papadimitriou, C. H., and K. Steiglitz. (1977) "On the Complexity of Local Search for the Traveling Salesman Problem". SIAM J. Commit.6, 78-83.

Revelle, C., D. Marks and J. C. Liebman (1970). "An Analysis of Private and Public Sector Facilities Location Models," Management Science 16, 12, 692-707.

Rosenkrantz, D. J . , R. E. Stearns, and P. M. Lewis

USING REPRODUCTIVE EVALUATION TO IMPROVE GENETIC SEARCH AND HEURISTIC DISCOVERY

convergence behavior can prevent the Genetic

Algorithm from discovering an acceptable solution.

Work toward a unified thermodynamic operator has

been undertaken to improve performance in ordering

problems such as the Travelling Salesman Problem.

The operator uses a model of genetic activity based on the Boltzmann distribution to allow control of convergence by a global temperature value as is done in Simulated Annealing. Introduction For analysis, the optimization tasks performed by the Genetic Algorithm (GA) can be grouped into two classes: value problems and order problems. Value problems are those problems, traditionally well suited to a GA representation, whose solutions can be naturally represented as a set of assignments of values to variables (in genetic terminology, the assignment of alleles to genes). Ordering problems are those problems, traditionally more difficult to represent, whose solutions are naturally represented as an ordered list of values (the assignment of genes to loci). Examples of problems in the GA literature which are naturally represented as ordering problems include the Travelling Salesman Problem (TSP), Job Scheduling, and Priority Assignment. Diversity of individuals in the population is required to find good solutions with the GA. A new operator is presented within the framework of this extension that unifies crossover, inversion, and mutation operators with concepts from Simulated Annealing. The operator uses a "temperature" parameter to provide control of the genetic diversity in the population. Diversity can be introduced by raising the temperature. The GA works to remove the diversity when the temperature is lowered. Ordering Problems Ordering problems can be represented in the GA by encoding them in a way that allows them to be treated as value problems. This approach has several drawbacks. The first difficulty is that the encoded form of the solution may not retain information when the genetic operators are applied. Second, low order schemata may no longer contain important information about the solution. Third, the encoded form may change the problem so that legal solutions are omitted or illegal solutions are included. Fourth the encoding may increase the size of the solution space which must be searched because representations of permutations are

inherently less dense than binary representations of the same length. Finally, the fitness of an encoded solution will typically be more expensive to compute because the individual must first be decoded. A natural extension to the GA along the lines suggested by Holland's inversion operator IRef. 1) allows the GA to be applied to ordering problems. Ordering problems can be accommodated more naturally by allowing the GA to compute the locus of each gene in addition to the allelles. The final order of the genes can then be interpreted directly as the desired permutation. This serves two purposes. In a value oriented problem it allows the GA to group the genes of a successful schema together. Second, in an ordering problem, the position of the gene in the individual can represent the ordering information required to solve the problem. In mixed problems the value and ordering mechanisms can operate together. However, this problem will be larger than usual because the solution space of this mixed problem will be the product of the solution spaces of each of the separate subproblems. As an example of this last case consider assigning school busses to bus stops. The route to be taken can be represented by the order of the stops in the individual and the bus assigned to each stop can be represented by the value of the gene. For instance, parent-1 in Figure 1 represents a route in which bus 1 stops first at St. Charles, then at Ventnor, and so on... PARENT-1 NAME ST CHARLES PARK PLACE VENTNOR ATLANTIC BOARDWALK VALUE BUS-1 BUS-2 BUS-1 BUS-2 BUS-3 PARENT-2 NAME ST CHARLES ATLANTIC VENTNOR PARK PLACE BOARKWALK VALUE BUS-1 BUS-3 BUS-3 BUS-i BUS-2 OFFSPRING OF CROS i SOVER AT POINT 1 NAME ST CHARLES PARK PLACE ATLANTIC VENTNOR BOARDWALK VALUE BUS-1 BUS-2 BUS-3 BUS-3 BUS-2 Figure 1. Crossover with mixed representation 116 This extended representation requires changes

to the genetic operators to allow them to operate on

individuals encoded in this way. Inversion can be

implemented in this context simply by allowing it to

invert the order of the genes in a substring of the

individual without changing their alleles. Crossover

can be extended in the following way: Select a

crossover point in parent #1; Transcribe allele/gene

pairs from parent #1 into the offspring up to the

crossover point in the order found in parent #I;

Complete the offspring by copying the remaining

allele/gene pairs from parent #2 in the order in

which they occur in parent #2 (Figure 1). Mutation

is unchanged because the inversion and crossover

defined above are sufficient to guarantee that no

permutation will be permenantly lost from the

population's potential search space. Implemented System
We have implemented the Genetic Algorithm on a

Symbolics Lisp Machine. The system uses the

representation for individuals described just above.

There are other aspects of the system that are

"non-standard" as well. These will be described

briefly below. These extensions were developed

before our work on the thermodynamic operator.

Because of their observed usefulness they have been

retained in the current work. In the standard GA each
individual's fitness is

evaluated by a single function. In contrast we allow

multiple independent fitness measures. Individuals

are chosen for breeding based on these measures with

each measure used in a proportion defined by the

analyst. One of the fitness measures is designated

"primary". This is the parameter that is really

being optimized. The other measures are "secondary"

and are used to promote the presence of desireable characteristics within the population. For instance, in the TSP, the primary fitness is the length of the tour. A secondary fitness we have used is based on the number of cities that are connected to their nearest neighbor. The rationale is that although simply connecting each city to its nearest neighbor (a greedy solution) does not produce an optimal route, the optimal route does contain many nearest neighbor connections. The "nearest-neighbor" secondary fitness is intended to promote the presence of such connections in the population. The secondary measures are typically invoked much less frequently than the primary measure, perhaps about 20% of the time. A copy of the current best individual as measured by each of the fitness measures is stored seperately from the main population. The "refresh" operator brings a copy of one of these current best individuals back into the population. This operator must be used sparingly (about 5%) because it can easily focus the population on a local minimum. Both the secondary fitness functions and the refresh operator are intended to increase the GA's performance by maintaining diversity yet retaining important information in a smaller population. We also use the heuristic crossover operator described by Greffenstette et al. [Ref 2]. Heuristic operators contain

domain knowledge and bring this knowledge to bear during a crossover operation. The heuristic for the TSP described in [Ref. 2] is to choose the next city to be visited from either parent, whichever is closer to the last city transcribed. This provides a powerful performance boost for the GA in solving travelling salesman problems. Later, we suggest briefly a way to incorporate heuristic operators into the proposed thermodynamic scheme. Convergence Rates Goldberg and Lingle [Ref. 3] introduce order schemata (o-schemata) as the ordering analog to the traditional allele schemata (called a-schemata in Ref. 3). In ordering problems the o-schemata are building blocks for optimal orderings just as the a-schemata are building blocks for optimal variable assignments in value problems. The hope has been that operators could be discovered that sift through the o-schemata allocating trials to them optimally as is done with a-schemata. However, the effect of the extended genetic operators on o-schemata is not immediately apparent. For instance, the crossover operator described above is much more destructive of o-schemata than a standard crossover is to a-schemata [Ref. 4). Put another way, the context sensitivity of o-schemata makes them fragile with respect to crossover. Consider the extended crossover described above. A crossover point is chosen randomly. Then all of the genes to the left of this point are transcribed from the first parent. These genes define o-schemata retained from the first parent. The remaining unused genes are then transcribed to the offspring from the second parent. However, since the genes that have already been used must be skipped, the section of the offspring to the right of the crossover point will not be identical to a section in either parent. In general these genes define new o-schemata not present in either parent. We call the expected number of schemata retained by an operator divided by the total number of schemata in an individual the retention ratio of the operator. We have estimated the retention ratios for the extended crossover and inversion operators described above. We find that as the length of the individual becomes very large, the o-schemata retention ratio for crossover approaches zero (most o-schemata are destroyed) while the o-schemata retention ratio for inversion approaches one (most o-schemata are retained). An example derivation of the o-schemata retention ratio for inversion is given in the appendix. While the destructive behavior of crossover could be tempered by introducing a "breeding" operator that just copies a selected individual (retention ratios = 1) and the activity of inversion increased by taking more than one cut point per individual it is not clear what the most desirable 117 retention ratios are or if they remain

the same over the duration of the GA optimization process. What is needed is a control parameter for schemata retention that allows the amount of information retained by genetic activity to be varied continuously. Simulated Annealing Simulated annealing [Ref. 5] provides just such a parameter, the temperature. In simulated annealing solutions are evaluated on the basis of their "energy". Disordered states (inefficient solutions) have a higher energy than ordered states (efficient solutions). A single best solution is maintained, rather than a population of solutions as in the genetic algorithm. The probability that a newly generated solution is accepted as the current best solution depends on the temperature and relative energies of the current best and new solution as follows: if <E n < E c) p = 1 if (E n > E c) p = ex Pl< E c " E n)/kT] (eq 1) where E n is the energy of the new solution and E £ is the energy of the current best solution. Conceptually, the system must pay a price in terms of energy to transition to a state that is higher in energy than the current state. The energy required is supplied by the heat present in the system. Boltzmann's equation relates the temperature to the available energy. The equation is probabalistic rather than deterministic since the ambient temperature represents the average of a large number of small volumes. Even at a low ambient temperature the local temperature in a small volume might be very high. Boltzmann's equation gives the probability that a given amount of energy is available at a given temperature. In simulated annealing the temperature is set to a high value initially and then lowered according to an annealing schedule provided by the analyst. There are reports that simulated annealing has enjoyed great success at the travelling salesman problem [Ref. 6]. These reports are difficult to assess, since performance numbers are not given. In any event, it would be desirable to include some aspects of simulated annealing within the genetic algorithm. In general framework of the the schemata retention operators (crossover, temperature parameter. genetic operators as section. The model the sense that it is particular we desire to tie behavior of the standard inversion and mutation) to a

This suggests a model for

described in the following

described is artificial in

motivated by a need to control schemata retention in

a computer program rather than by biological or

physical processes. Thermodynamic Operator We consider a crossover operation between two

parent individuals that have been selected from the population with probability proportional to their fitness. An offspring individual is constructed by transcribing the genes from the parents in such a way that o-schemata and a-schemata are partially preserved. Transcription begins with the first gene of one of the parents. We imagine that it is energetically favorable to continue transcription from the same parent; it costs energy to switch transcription to the other parent. The energy available in the environment follows a Boltzmann distribution. If we call the energy at which transcription will switch to the other parent Q (the crossover threshold energy) then a switch will occur just when the energy available locally (which depends on the temperature) exceeds 6 • This occurs with probability exp [-6,/kT] (eq 2) where k is an arbitrarily chosen scaling constant. We assume k = 1. Transcription continues from the second parent until the crossover threshold energy is again exceeded. At high temperatures, transcription might switch back and forth between the parents many times. At low temperatures transcription might not switch at all resulting in a direct copy of the parent individual. Inversion is modelled similarly. We imagine that the strings selected from each parent stand a chance of being rotated as they are transported from the parent to the offspring (see figure 2). Again, it costs energy to perform this rotation and this energy is supplied by the heat available locally. Thus, the probabilty of inversion is exp [-e./kT] where 9. is the inversion threshold energy. In some domains it may be desirable to model the fact that it requires more energy to rotate a longer string than to rotate shorter one. Due to the rotational symmetry in the travelling salesman problem we ignored this issue. That is, the distance from city-i to city-j is the same as the distance from city-j to city-i so that in this case, o-schemata wholly contained within the rotated string are not affected by the operation. In other cases, for instance bin packing, packing item-i before item-j will almost always result in a different fitness than packing item-j before item-i. 118 OFFSPRING Figure 2. A s t r ing may be rotated STFIING BETWEEN CROSSOVER POINTS PARENT 119 Figure 4. Temperature vs Time for a typical annealing schedule Figure 3. Overview of a system to implement the GA The heuristic employed in [Ref. 2] for the travelling salesman problem is to choose the next city to be visited

from the next available gene in either parent, whichever is closer to the last city transcribed. One could imagine, instead, evaluating the relative energies of the two choices as in simulated annealing and then using (eq 1) to make a choice. Thus, the less favorable alternative is more likely to be chosen at higher temperatures than at lower ones. This at least ameliorates the deterministic quality of the heuristic operator. An implementation difficulty for this idea is that there does not seem to be a way to avoid an expensive exponentiation operation at each gene during transcription. Annealing Schedule The overall performance of the GA using the thermodynamic operator is sensitive to the annealing schedule chosen by the analyst. The general strategy we have used is to start the system at a high temperature, drop the temperature fairly rapidly to a moderate range and then descend more slowly to a very low temperature. The process is then repeated with a slightly lower initial temperature. This can continue until the initial temperature becomes too close to the final temperature. Then the process can be repeated again starting from a high temperature (figure 4). T E M P E R A T U R E TIME

Thus inverting a longer string in the bin packing

case should require more energy than inverting a

shorter string. Finally, mutation (of the allele) is treated

similarly. We imagine that as each allele is

transcribed there is a probability that its value

will change. The probability of nutation is just exp [-θ 1 /kT] Thus a thermodynamic operator is specified by a

triple: [θ c, θ i ,θ m]. The overall system

temperature is varied according to an "annealing"

schedule chosen by the analyst. As the temperature

approaches' 2ero, the probability that any of the

thresholds will be exceeded also approaches zero,

genetic activity ceases, and the schemata retention

ratio approaches 1. As the temperature approaches

infinity, schemata are retained only by chance and the algorithm becomes a random search. The operator therefore provides a retention ratio for o-schemata that is continuously variable from 0 to 1.

Implemented Operators We have implemented the unified thermodynamic operator and use it in place of the basic genetic operators. Since the exponentiation operation in (eq 2) is computationally expensive, we calculate the expected number of genes to be transcribed at the current temperature before a crossover event occurs. This quantity is given by a geometric distribution [Ref. 7): N = [In U/ In (1 - eθ/kT)] where U is a uniform deviate in the interval [0,1]. The calculation is the same for crossover, inversion and mutation. These quantities should be recalculated if the temperature is raised. In the thermodynamic framework refresh is implemented by allowing the operator to be defined by a 4-tuple (θ c , θ i , θ m , r) where 0<r<l. Normally, individuals are selected as parents with probability proportional to their fitness. The parameter, r, specifies the probability of using the saved copy of the current best individual instead. When there are secondary fitness measures one of the current best individuals is selected randomly. At

high temperatures this individual is altered

significantly through crossover with the second

parent before being returned to the population but

at low temperatures it tends to be copied directly

into the population. An overview of the system is

shown in figure 3. Although we have not done so, it would seem

that the heuristic operators could also be placed in

the thermodynamic framework. (We use a heuristic

operator which is seperate from the thermodynamic

operator.) 3EST POP REFRESH SELECTION JURRENT POP THERMODYNAMIC CROSSOVER NtW N 3EST?, 20.000 TRIALS 45,439 TRIALS FITNESS = 0.9155 FITNESS = 0.9282 Figure 7. A 100 c i ty problem 120 RANDOM TOUR 2500 TRIALS POPULATION SIZE =10 FITNESS = 0.8976 50 CITIES 5000 TRIALS 10,310 TRIALS FITNESS = 0.9367 FITNESS = 0.9443 Figure 6. A 50 c i ty problem RANDOM TOUR 2500 TRIALS POPULATION SIZE = 12 FITNESS = 0.7567 100 CITIES The spec i f ic annealing schedule used for a 100 c i t y problem i s implemented by the routine shown in Figure 5. Two individuals are created and evaluated between each c a l l to t h i s update rout ine . (Defun update-temp () (cond ((> *temp* 115) (decf *temp* 1.0)) ((> *temp* 100) (decf *temp* 0.2)) ((> *temp* 50) (decf *temp* 0.1)) ((> *temp* 36) (decf *temp* 0.05)) (t (progl (setq *temp* *reset-temp*) (i f (< *reset-temp* 50) (setq *reset-temp* 150) (decf *reset-temp* 10)))) Figure 5. A LISP routine to implement an annealing schedule Test Cases We have run many experiments with travelling salesman problems of various sizes. Some sample cases are exhibited in figures 5, 6, and 7. Ue have found that the use of thermodynamic crossover in conjunction with heuristic crossover provides a marked improvement over the use of the extended crossover operator alone or heuristic crossover alone. We have also found that thermodynamic crossover allows the reduction of the population to sizes much smaller than those indicated by Goldberg and Lingle [Ref. 31. For instance, we use a population size of 12 - 15 individuals for a 200 city problem. Other parameter settings used in the 100 city problem are as follows. G

: 375.0 c 6.: 140.0 6 : not applicable m refresh: .07 initial temperature: 250.0 The annealing schedule shown in figure 5 was used. The heuristic crossover operator was invoked 255! of the time and thermodynamic crossover 75Z. A population of size 12 was used. In the following figures "Trials" is the number of individuals that have been evaluated. The fitness measure is in terms of the theoretical optimum value derived in (Ref. 8] and referenced in [Ref. 9J. The fitness quantity used is the optimum tour length divided by the actual tour length. Thus a fitness of .95 is about 5X above the theoretically predicted minimum tour. These results compare favorably with results presented by other researchers using the GA on the TSP, [Ref. 21 for example, both in terms of the required number fitness evaluations and the resulting route length. 121

RANDOM TOUR 2500 TRIALS

POPULATION SIZE = 12 FITNESS = 0.5607

200 CITIES

35,000 TRIALS ~ '" 70,676 TRIALS

FITNESS = 0.95037 FITNESS = 0.9656 Figure 8. A 200 city problem Conclusion Although the unified Thermodynamic Operator is

not directly motivated by a theory of genetics or

thermodynamics it provides an explicit control over

population convergence. Empirically, it has been

shown to increase performance through the use of an

annealing schedule, a concept borrowed from

Simulated Annealing. Analysis of the thermodynamic

operator and a better understanding of the

relationship between schemata retention and optimal

convergence rates is expected to lead to even better

performance by customizing the annealing schedule to

the requirements of specific applications. Appendix

0-Schemata Retention Ratio for Inversion The inversion operator on an individual of

length L selects two unique points A and B. The

order of the genes between these points is reversed.

To calculate the o-schemata retention ratio (the

fraction of o-schemata retained by a typical

application of the inversion operator) we first

calculate the expected number of o-schemata retained

and then divide by 2 , the total number of

o-schemata originally represented in the individual. To calculate the expected number retained, E,

characteristic description for a set of strings 1-allele schemata), are

Grefenstette, JJ . [1986] Optimization of

Control Parameters for Genetic Algorithms. IEEE Transactions on Systems, Man, and

Cybernetics. Vol. SMC-16, No. 1,

January/February. Holland, J.H. [1986] Escaping Brittleness: The Possibilities of General Purpose Algorithms Applied to Parallel Rule-Based Systems. In Machine Learning: An Artificial Intelligence Approach Vol. 2 by Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (Eds.) Morgan Kaufman. Holland, J.H, Reitman, J.S. [1978] Cognitive Systems based on adaptive algorithms. In Pattern-Directed Inference Systems by Waterman, D.A., Hayes-Roth, F. (Eds.), Academic Press. Lebowitz, M. [1986] UNIMEM, a general learning system: An overview. Proceedings of ECAI-86, Brighton, England. McGregor, D.R., Malone, J.R. [1982a] Generic Associative Hardware. Its Impact on Database Systems. Proceedings IEE Colloquium on Associative Methods & Database Engines, May 1982. McGregor, D.R, Malone, J.R. [1982b] Generic Associative Memory, G.B. Patent No. 8236084. McGregor, D.R., Malone, J.R. [1983] The Fact System - a Hardware-oriented approach. In DBMS A Technical Comparison - State of the Art Report. Maidenhead; Pergamon Infotech pp 99-112. Michalski, R.S. [1983] A Theory and Methodology of Inductive Learning. Artificial Intelligence, Vol. 20 pp. 111-161. Oosthuizen, G.D. [1986] A Paradigm for Automatic Learning. Internal FACT 24/86. University of Strathclyde, Scotland. Oosthuizen, G.D., McGregor, D.R., Henning, M., Renfrew, C. [1987a] Parallel Network Architectures for Large Scale Knowledge Based Systems. Proceedings of First Workshop of the British Special Interest Group on Knowledge Manipulating Engines (SIGKME). Reading, England. January 1987. Oosthuizen, G.D., McGregor, D.R., Malone, J.R. [1987b] The Use of a Simple Connectionist Architecture for Matching and Learning. Internal Report FACT 2/87. University of Strathclyde, Scotland. Schrodt, P.A. [1986] Predicting international events. BYTE, November. 139

PARALLEL IMPLEMENTATION OF GENETIC ALGORITHMS IN A CLASSIFIER SYSTEM George G. Robertson Thinking Machines Corporation Abstract

29. R u m e l h a r t , D.E.. Hinton. G.H.. and Williams R.J.. Learning Internal Representations by Error Propagation. 1CS R e p o r t 8506, UC San Diego, September 1 9 8 5 .

30. Sejnowski. T.J.. NETtalk: A Parallel Net-work that Learns to Read Aloud. Johns Hopkins Univ. Electrical Engineering and Computer Sci-ence Technical Report JHU/EECS-86/01, Jan-uary, 1986.

31. Sejnowski, T.J., Keinker. P.K.. and Hinton, G.E., Symmetry Groups with Hidden Units: Beyond the Perceptron, Physica D (in press).

32. Steele, G.L., Common Lisp: The Language. Dig-ital Press, Massachusetts. 1984.

33. Thinking Machines Corporation, Introduction to Data Level Parallelism, Thinking Machines Technical Report, April, 1986.

34. Wagner, H.M., Principles of Operations Re-search. Prentice-Hall, New Jersey, 1969.

35. Wilcox, B., Reflections on Building Two Go Pro-grams. SIGART Newsletter, October 1985.

30. Wilson. SAW, Classifier System Learning of a Boolean Function. Rowland Institute for Science Research Memo RIS No. 27r, February, 1986.

37. Wilson. S.W., Knowledge Growth in an Artificial Animal, in J.J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algo-rithms and their Applications, Carnegie-Mellon Univ., Pittsburgh, Pa., July, 1985. 147 PUNCTUATED EQUILIBRIA: A PARALLEL GENETIC ALGORITHM by J. P. Cohoon, S. U. Hegde, W. N. Martin, D. Richards Department of Computer Science University of Virginia Charlottesville, Virginia 22903 ABSTRACT A distributed formulation of the genetic algorithm paradigm is proposed and experimentally analyzed. Our formulation is based in part on two principles of the paleontological theory of punctuated equilibria—allopatric speciation and stasis. Allopatric speciation involves the rapid evolution of new species after being geographically separated.

Stasis implies that after equilibria is reached in an environment there is little drift in genetic composition. We applied the formulation to the Optimal Linear Arrangement problem. In our experiments, the result was more than just a hardware acceleration, rather better solutions were obtained with less total work. INTRODUCTION The genetic algorithm paradigm has been previously proposed to generate solutions to a wide range of problems [HOLL75]. In particular, several optimization problems have been investigated. These include control systems [GOLD83], function optimization [BETH81], and combinatorial problems [COHO86, DAVI85, FOUR85, GOLD85, GREF85, SMIT85]. In all cases, serial implementations have been proposed. We will argue that there is an effective parallel realization of the genetic algorithms approach based on what evolution theorists call "punctuated equilibria." We propose a parallel implementation and present empirical evidence of its effectiveness on a combinatorial optimization problem. While the genetic algorithm (GA) approach is easily understood, it would be difficult to glean a canonical "pseudo-code" version from published accounts. Various implementations differ and many "obvious" design decisions are omitted. In all cases, a population of solutions to the problem at hand is maintained and successive "generations" are produced by manipulating the previous generation. The population is typically kept at a fixed size. Most new solutions are formed by merging two previous ones; this is done with a "crossover" operator and suitable encodings of the solutions. Some new solutions are simply modifications of previous ones, using a "mutation" operator. Successive generations are produced with new solutions replacing some of the older ones. An ad hoc termination condition is used and the best remaining solution (or the best ever seen) is reported. A solution is evaluated with respect to its "fitness," and of course we prefer that the most fit survive. There are two mechanisms for differential success. First, the better fit solutions are more likely to crossover, and hence propagate. Second, the less fit solutions are more likely to be replaced. It is important to realize that the GA approach is fundamentally different from, say, simulated annealing [KIRK83] which follows the "traectory"of a single solution to a local maximum of the fitness function. With GA there are many solutions to consider and the crossover operation is so chaotic that there is no simple notion of trajectory. How can parallelism be used with the GA approach? Initially it seems clear that t h e process i s inherently sequential. Each generation must be produced before it can be used as the basis for the following generation; it is antithetical

to the evolutionary scheme to jump forward. A simple use of parallelism is the simultaneous production of candidates for the next generation. For example, pairs of solutions could be crossed-over in parallel, along with the selection and mutation of other solutions. But algorithmic issues remain to be resolved. How are t h e "parents" probabilistically selected? How are the solutions that are to be replaced chosen? The simple answers to these questions, which require global information, suggest the use of shared-memory architectures. Note that this sort of parallelism does not make any fundamental contribution to the GA approach; it can be viewed simply as a "hardware accelerator". We restrict our interest to the s tudy of paralled algorithms for a distributed processor system without shared memory. Our reasons are threefold. First, we have access to such a system. Second, the extension of our results to massively parallel machines will be quite natural. In such a machine, the cost of connecting and distributing data is an important component in the analysis of the algori thms. We assume that the interconnection network i s space, and hence communication between distant processors is expensive. Third, as implied above, we are interested in developing more than just a hardware accelerator. Rather we desire a distributed formulation that gives better solutions with less total work. We feel the most natural way to distribute a genetic algorithm over the processors is to partition the population of solutions and assign one subset of the population to the local memory of each processor. Consider a straightforward implementation of t h e GA a p p r o a c h . In order to probabilistically select parents for the crossover operation, 148

global information about the (relative) fitnesses must be

used. This implies an often performed phase of data

collection, processing, and broadcasting. Further, extensive

data movement is required to crossover two randomly

selected solutions that are on distant processors. Considerations such as the above led us to question the

wisdom of using the GA approach as it is typically

presented. There are many ad hoc methods for bypassing

these difficulties. For example, continuing the above

examples, knowledge of the global fitness distribution can be

approximated. Further, steps can be taken to artificially

reduce the diameter of related computations. Instead we

found a simple model that naturally maps GA onto a

distributed computer system. It is drawn from the theory of

punctuated equilibria, discussed in the next section. We chose to do our initial study using the Optimal

Linear Arrangement problem (OLA). It is an NP-complete

combinatorial optimization problem [GARE79]. We selected

it due to the practical interest in such placement problems, as

well as its simple presentation. There are m objects and m

positions, where the positions are arranged linearly and are

separated by unit distances. For each pair of objects i and j

there is a cost c i j . We need to find an mapping p, where

object i is assigned to position p(i), that minimizes the

objective function $\sum_{i<j} c_{ij} \, | p(i) - p(j) |$. (1)

We note that OLA is related to the ubiquitous traveling

salesman problem; they are both instances of the quadratic

assignment problem.

PUNCTUATED EQUILIBRIA

N. Eldredge and S. J. Gould [ELDR72] presented the theory

of punctuated equilibria (PE) to resolve certain

paleontological dilemmas in the geological record. While

the extent to which PE is needed to explain the data is

hotly

debated [ELDR85], we have found it to be an important

model for understanding distributed evolutionary processes.

PE is based on two principles: allopatric speciation and

stasis. Allopatric speciation involves the rapid evolution of

new species after being geographically separated. The

scenario involves a small subpopulation of a species,

"peripheral isolates," becoming segregated into a new

environment. By using latent genetic material or new

mutations this subpopulation may survive and flourish in its

environment. A single species may give rise to many

peripheral isolates. Stasis, or stability, of a species is simply the notion of

lack of change. (This directly challenges phyletic

gradualism.) It implies that after equilibria is reached in an

environment there is very little drift away from the genetic

composition of a species. The motivation is that

"sympatric" speciation (differentiation in the same

environment) is difficult since small changes can not

compete w i t h t h e "gene f low" of current species.

Ideally a species would persist until its environment changed

(or it would drift very little). It is instructive to define "species" in a way that relates to the concept of a solution in the GA approach. We adapt an old idea of S. Wright [WRIG32] that introduces the concept of the "adaptive landscape," which is analogous to the fitness "surface" over the space of solutions. Consider a peak of

the landscape that has been discovered and populated by a subset of the gene pool. That subset (perhaps with nearby subsets) corresponds to a species. There can be many "species" in a given environment, some so distant that their mutual offspring are not adapted to the environment. The difficult question is how can a species, as a whole, leave its "niche" to migrate to an even higher peak. The concept of stasis emphasizes the problem. PE stresses that a powerful method for generating new species is to thrust an old species into a new environment, that is, a new adaptive landscape, where change is beneficial and rewarded. For this reason we should expect a GA approach based on PE should perform better than the typical single environment scheme. What are the implications for the GA approach? If the "environment" is unchanging then equilibrium should be rapidly attained. The resulting equivalence classes of similar solutions would correspond to species. It is possible that the highest peaks have been unexplored. Typically, when GA is used the mutation and crossover operations are relied on to eventually find the other peaks. PE indicates that a more diverse exploration of the adaptive landscape could be achieved by allopatric speciation of peripheral isolates. Therefore, subpopulations must be segregated into environments that are somehow different. Two different schemes for changing the environment are suggested. Suppose fitness is a multi-objective function. Various low-order approximations to the true fitness could be tried at different times and places. We will not explore this further here. The second scheme simply changes the environment by throwing together previously geographically separated species. We feel that the combination of new competitors and a new gene pool would cause the desired allopatric speciation. Further, we will define fitness so that it is relative to the current local population. So a new combination of competitors will alter the fitness measure. This scheme is used in the work presented here. GENETIC ALGORITHMS WITH PUNCTUATED EQUILIBRIA Our basic model of parallel genetic algorithms assigns a set of n solutions to each of N processors, for a total population of size nxN. The set assigned to each processor is its subpopulation. (It is a simple extension to the model to allow different and time-varying population sizes. Other extensions are discussed in Section 6.) The processors are connected by a sparse interconnection network. In practice we might expect a conventional topology to be used, such as a mesh or a hypercube, but at present the choice of topology is not considered to be important. The network should have high connectivity and small diameter to ensure adequate "mixing" as time progresses. The overall structure of our

approach is seen in Figure 1. There are E major iterations called epochs. During an 149

epoch each processor, disjointly and in parallel, executes the

genetic algorithm on its subpopulation. Theoretically each

processor continues until it reaches equilibrium. Since we

know of no adequate stopping criteria we have used a fixed

number, G, of generations per epoch. This considerably

simplifies the problem of "synchronizing" the processors,

since each processor should be completed at nearly the same

time. After each processor has stopped there is a phase

during which each processor copies randomly selected

subsets of its population to neighboring processors. Each

processor now has acquired a surplus of solutions and must

probabilistically select a set of n solutions to survive to be its

initial subpopulation at the beginning of the next epoch.

(The selection process is the same as the adjustment

procedure used by GA proper.) The relationship to PE should be clear. Each processor

corresponds to a disjoint "environment" (as characterized

by the mix of solutions residing in it). After G generations

we expect to see the emergence of some very fit species. (It

is not necessary or even desirable to choose G so large that

only one "species" survives. Diversity must be

maintained.) Then a "catastrophe" occurs and the

environments change. This is simulated by having

representatives of geographically adjacent environments

regroup to form the new environments. By varying the

amount of redistribution, that is, S = |Sij|, we can control the

amount of disruption. There can be two types of probabilistic selection used

here. The fitness of each element of a population is used for

selection, where the probability of selecting an element is

proportional to its fitness. When there is repeated selection

from the same population it can be done either with

replacement or without replacement The decision should be

based on both analogies with the natural genetic model and

goals for efficiently driving the optimization process. The

selection of each final (end of epoch) subpopulation is done initialize for E iterations do parfor each processor i do run GA for G generations endfor parfor each processor i do for each neighbor j of i do send a set of solutions, S ij , from i to j endfor endfor parfor each processor i do select an n element subpopulation endfor endfor

Figure 1 — The parallel genetic algorithm with punctuated

equilibria. without replacement; the good solutions will only "propagate" at the beginning of the next epoch. (The selection of each S i j i s not done probailistically.A "random", i.e. using a uniform distribution, selection is used to simulate the randomness of environment shifts.) We present our interpretation/implementation of the GA code each processor uses in Figure 2. The crossover rate, $0 \leq C \leq 1$, determines how many new offspring are produced during each generation. "Parents" are chosen probabilistically with replacement The crossover itself, and other details, are discussed below. Our crossover produces one offspring from two parents. The fitnesses are recalculated, relative to the new larger population. Then, probabilistically without replacement, the next population

is selected. Finally, (uniform) random elements are mutated. The mutation rate, 0 ≤ M ≤ l , determines how many mutations altogether are performed. IMPLEMENTATION DETAILS The problem we studied, OLA, is a placement problem. Hence a solution must encode a mapping p from objects to positions. Since we may assume both objects and positions are numbered 1,2,..., m, the mapping p is just a permutation. In fact, throughout we use the inverse mapping, from positions to objects, as the basis for our encodings. There are many ways to encode permutations but it is not at all clear which, if any, are suitable for the GA approach. Note that it is desirable to preserve adjacencies within groups of objects during crossover. Several representations and crossovers have been proposed for related problems, e.g. the traveling salesman problem. Inversion vectors ("ordinal representations") are the most obvious choice since they allow "typical" crossovers (as in [HOLL75]). However the use of such crossovers with inversion vectors is quite undesirable [GREF85] since it breaks up groups of adjacent objects. Goldberg and Lingle [GOLD85] gave a "partially mapped crossover" that uses a straightforward array representation of the (inverse) mapping, where the i th entry is j if the j th object is in the i t h postion Briefly, their crossover copied a contiguous portion of one parent into the offspring, while for G iterations do for nxC iterations do select two solutions crossover those solutions add offspring to subpopulation endfor calculate fitnesses select a population of n elements generate nxM random mutations endfor Figure 2 — The genetic algorithm used within an epoch at each processor. 150

having the other parent copy over as many other positions as

possible. Smith [SMTT85] proposed a "modified crossover"

for the array representation. A random division point is

selected and the first "half of one parent is copied to the

offspring. The remainder of the offspring's array is filled

with unused objects, while preserving their relative order

within the other parent. For example, parents [7 1 3 5 6 2 4]

and [3 4 2 7 1 5 6] w i t h the division point after the third

position produce the offspring [7 1 3 4 2 5 6] . We essentially used this representation and crossover but allowed the first or the second " h a l f " of the first parent to be used. By arguments analogous to those of Goldberg and Lingle [GOLD85], we can argue that our approach has the desirable "schema-preserving" property that the GA approach exploits. However it is an open problem to give a theoretically compelling proof of that property. In any event, it is clear that our scheme tends to preserve blocks of adjacent objects. The mutate operator was selected next. We felt that the mutations should not be too disruptive; if most adjacencies were broken then with near certainty the mutation would be immediately lost. We chose to use "inversion," the reversal of a contiguous block within the array representation. The beginning of the block was randomly selected. The length of the block was randomly chosen from an exponential distribution with mean μ . W e typically kept μ small to inhibit disruption. The nature of the OLA problem encourages inversion as opposed to pairwise interchanges, which do not involve block moves. How should fitness be calculated? With any minimization problem, such as OLA, the scores of the solutions should decrease over time. The score is the value of the objective function. Two simple fitness functions

suggest themselves. First, the fitness could be inversely

related to the score; this could cause excessive compression

of the range of fitnesses. Second, the fitness could be a

constant minus the score. The constant must be large

enough to ensure all fitnesses are positive (since they are

used in the selection process) and not too large
(effectively

causing compression). If such a constant was optimal

initially, it would become a poor choice near equilibria.
For

these reasons we used a time-varying "normalized" fitness.
We chose our fitness to be a function of all the scores in

the current population. We have empirically found that

randomly generated solutions to the OLA problem have

scores that are "normally distributed" (i.e., have a bell-

shaped curve), with virtually every solution within 1

standard deviation (s.d.) of the mean, and no solutions were

found more than 3 s.d.'s away. For related evidence see

[COH087, WHIT84]. Therefore we used Our fitness measure
has several advantages. It is somewhat problem
independent, so mat we can reasonably compare very
different instances. It also tends to control the effect
of a few "outliers" on the population. A disadvantage is
that it is expensive to calculate and it needs to be
recomputed at regular intervals. An approximation scheme
can be used where new fitnesses are calculated according
to the current mean and variance. The other elements would
not have their fitnesses recalculated, unless they were
otherwise encountered, but the pseudo-normalization renders
them comparable. EMPIRICAL RESULTS We performed several
experiments to determine if our parallel genetic algorithm
is an effective approach. The efficacy of any GA approach
is determined by the design variables. Our initial
experiments, reported here, have been made with the
Optimal Linear Arrangement (OLA) problem as the base case.

This permutation problem has a raw score (Eq. 1) that the system is to minimize for a given cost matrix. The system uses a fitness measure (Eq. 2) in selecting the elements for crossover and in determining the survivors for each generation. Remember that for both of those selection processes within a subpopulation the fitness is judged relative to that subpopulation, and that the fitness is used in a probabilistic manner. Our current implementation is a sequential simulation of the parallel genetic algorithm with punctuated equilibria. It operates with an arbitrary configuration of the N subpopulations. Presently we are investigating "mesh" and "hyper-cube" connection topologies. Although other configurations will be analyzed, the hyper-cube topology is of particular importance to us. We plan to obtain "real" performance measures on the hyper-cube multiprocessor at the University of Virginia. In most of the initial studies described below, a mesh configuration is used with N = 4, i.e., each subpopulation being able to "communicate" during the inter-epoch transition with two other subpopulations. For each experiment the number of epochs, E, is given along with the number of generations per epoch, G, and the end-of-generation subpopulation size, n. Thus, over the course of a single example the parallel system will create NxExG generations. If we set N and E to one, then we have a "standard" sequential GA creating G generations. While the sequential GA has a single evolitionary time line,the parallel algorithm has multiple, interrelated evolutionary time lines. The "interrelated" qualification is quite important because the paralled system does more than just create N divergent time lines. As w i t h the sequential GA, one cannot say, a priori, how many distinct individuals, i.e., possible problem solutions, a particular example run of our system will examine. For our purpose here, we will use NxExGxn to be an indicator of t h e total number of our created during the experiment. The remaining design variables of importance are C, the crossover rate, M, the mutation rate, S, the size of the redistribution set, α the fitness scale factor, and μ, the mean lenght of the mutation block. 151 Copyrighted Material

where μ s is the mean of the score,σ i s the s.d.,and α i s a

small constant parameter. Note that in practice we expect

0 < fitness (x) < 1; we use clipping to ensure it is positive.

Near equilibrium the scores will not be normally distributed

because the contributions from most mutations and many

crossovers will almost certainly be below the mean, biasing

the distribution. fitness(x) = (μ s -score (x)) + ασ
2ασ (2) Results For Three Problem Instances l C M S μ
n α N E G ŝ s ∗ s 1 0.50 0.05 15 3 80 3.00 4 2
50 9600 9600 9600 1 400 9600 9600 4 100 9600 9600 2
0.50 0.05 15 3 80 3.00 4 6 50 19200 19200 19200 1
1200 19200 19200 4 300 19200 21757 3 0.50 0.05 15 3 80
3.00 4 6 50 15510 15551 15450 1 1200 15540 15622 4
300 15540 15596 Table 1 The Effects Of Changing The
Design Variables C M S μ n a N E G ŝ s s∗ 0.50 0.05
15 3 80 3.00 4 6 50 15225 15318 15210 0.80 0.20 40
12 50 1.50 0.50 12 25 15240 15270 15240 15270
15225 15210 15210 15210 15315 15311 15343 15360
15371 15258 15225 15266 Table 2 algorithm would
require about four (N) times the amount of "wall clock"
time as the parallel genetic algorithm with punctuated
equilibria. The third row shows the results from a simple
parallel genetic algorithm that just used four (N)
independent populations without communication. Here each
example run was derived by setting N to four and E to one,
and then, at the end of the parallel operation, selecting
the best overall from the best of the four populations.
For each instance, we kept NxExGxn constant over all three
rows. This product is indicative of the total number of
OLA solutions examined by the system. By keeping the
product constant we assume that approximately the same
amount of total computation is required. For these initial
studies we have considered "artificial" OLA examples,
which allow easy determination of the optimal score. While
the examples are contrived, they exhibit natural
clustering patterns. In all cases, the costs were chosen
so that the identity permutation produced the optimal
score; the only other optimal permutations were simple
perturbations of the identity mapping. (Of course this
does not make the problem any easier.) We used three types
of problems, i.e., cost matrices, in our experiments.
Tables 1 and 2 present the results and the settings used
to derive those results. In those tables the quantity, s ,
is the theoretical optimal OLA score (as opposed to the
fitness measure); s is the average of the best OLA score
from each example with the specified settings; and ŝ is
the score of the single best solution created during the
example. In the current simulations a single random number
process is used, allowing multiple examples to be
generated for the same design variable setting by changing

a single "seed." In the discussion below the term "average" will indicate that several examples with the same settings and inputs, but with different seeds, have been run and the resulting measures averaged. In all cases reported here, the average is taken over a minimum of four runs. Table 1 is broken into three instances, with three rows for each instance. The first row shows the results from running the parallel genetic algorithm with punctuated equilibria, i.e., independent subpopulations with communication. For these results a four node mesh configuration was used. The second row shows the results from a sequential genetic algorithm. That algorithm was derived by setting N and E t o one, i.e., one population creating G generations. For a comparable uniprocessors, this 152 The first type of problem instance, with unique optima,

has a cost matrix of the following form: 0 A B C D 0 0 0 0 A 0 A B C D 0 0 0 B A 0 A B C D 0 0 C B A O A B C D O C 1 (9)= D C B A O A B C D O D C B A O A B C 0 0 D C B A 0 A B 0 0 0 D C B A 0 A O O O O D C B A O

where m = 9 and A>>B>>C>>D≥0. We belive the

solution spaces for such problems instances to be "convex"

in some sense, and therefore "easy." Instance one (l= 1) of

Table 1 used C 1 (9),with A = 1000, B = 100, C = 10, and

D = 1. Note that, for this instance, the parallel genetic

algorithm with punctuated equilibria found the optimum

solution in each example, as indicated by s= s ∗. The second problem type was slightly more complex.

We increased m to 18 and created a cost matrix by

embedding two independent 9-element orderings as given by

the following cost matrix: C 1 (9) 0 0 C 1 (9) C 2 (18) = C 3 (9) =

Note that two groups of 9 are uncoupled and that this is

tantamount to solving two disjoint problems. A, B, C,

and D in C 2 (1 8) were as above. The setting and result for

this cost matrix are shown as instance two (l = 2)in Table 1. The third type of problem incorporated further

complexity by embedding interrelated blocks of three

elements each. For nine elements the resulting cost matrix

would be 0 A A C C C C C C A 0 A C C C C C C A A 0 B C C C C C C B 0 A A C C C C C C A 0 A C C C C C C A A 0 B C C C C C C C B 0 A A C C C C C C A 0 A C C C C C C A A 0

We assume A > B > C. The intra-block cost, A, causes

primary clustering, and the inter-blocks.The other costs,

blocks, B, forces an ordering of the blocks. The other costs,

the C's, tend to "flatten" the search space by making all

permutations have similar scores. This cost matrix pattern

was extended to C 3 (18), i.e., eighteen object comprised of

six blocks, and we let A = 50, B = 30, and C = 15. The

results are shown as instance three (l =3) in Table 1. The results shown in Table 2 are presented to indicate

the effects of changing individual design variables. The

Description of Experiments

50 10 5 1

0.5 0 25 50 75 100 generations 1 node 2 nodes 4
nodes 6 nodes 16nodcs 500 100 50 10 0 25 50 75
100 generations 1 node 2 nodes 4. nodes 8 nodes
16nodes 1 node 2 nodes 4 nodes 8 nodes 16 nodes 5010
5 1

0.5 0 25 50 75 100 generations Figure 4a. Function fl
Figure 4b. Function f2 Figure 4c. Function f3 Figure 4d.
Function f5 Figure 4. Offline Average Performances 160
10t nodes 2 nodes 4 nodes 8 nodes 16 nodes 5 1 0.50
25 50 75 100 generations 1 nodes J nodes 4 nodes 8
nodes 16nodes 100 10 50 5 0 25 50 75 100
generations 1 node 2 nodes 4 nodes 8 nodes 16nodes
0.5 1 5 0.1 0.05 0 25 50 75 100 0 0.5
generations 25 50 generations 75 100 50 105 1 •1)
node 2 nodes 4 nodes 8 nodes 16nodes

0.5

0.1

0.05

0.01

0.005-4 25 50 generations 75 100 0.5-4 0.1 0.050.01
0.005 0.001 25 50 generations 75 100 F i g u r e 5 a
. F u n c t i o n f l F i g u r e 5 b . F u n c t i o n f
2

10

1 25 50 generations 75 F i g u r e 5 c . F u n c t i o
n f 3 100 168 2 25 50 generations 75 100 F i g u r
e 5 d . F u n c t i o n f 5 F i g u r e 5 . O f f l i n e
B e s t P e r f o r m a n c e s 1 6 1

1 1 nodes 2 nodes 4 nodes 8 nodes 16nodes 0 0 1
nodes 2 nodes 4 nodes 8 nodes 16nodes

5 1 nodes 2 nodes' 4 nodes 8 nodes 16nodes

0.5 0 0 1 4 1 nodes 2 nodes 4 nodes 8nodes 16nodes
GENETIC LEARNING PROCEDURES IN DISTRIBUTED ENVIRONMENTS
Adrian V. Sannier II Research Assistant Erik D. Goodman
Professor and Director A.H. Case Center for Computer-Aided

Engineering and Manufacturing Michigan State University, East Lansing, MI 48823 ABSTRACT This paper introduces a strategy for fostering the develop-ment of hierarchically organized distributed systems which uses a genetic algorithm [1] to manipulate indepen-dent, computationally limited units that work toward a, com-mon goal. The central thesis of this work is that impor-tant characteristics of the hierarchical structure of living systems can be duplicated by applying idealized genetic operators to a functionally interacting population of encoded programs. The models and results we present in this paper suggest that the introduction of functional interaction between distributed units can promote the development of a genome capable of producing a set of independent, differentiated units and implicitly organizing them into a coherent and coordinated distributed system. The results presented here are available in more complete form in [2]. AN IDEALIZED DISTRIBUTED GENETIC SYSTEM The class of systems we have attempted to model emerges from consideration of macro-organisms composed of vast numbers of interacting, distributed units. During ontogeny, a single genetic program, or genome, represented in several strands of DNA, replicates itself, more or less exactly, millions of times. The cells containing these repli-cas of the original program do not all perform alike, howev-er. Instead, they differentiate into many types, each of which performs a specialized function. None of the individu-al units operates on the scale of the macro-organism, yet together the operations which these differentiated cells perform are implicitly coordinated to produce the behavior of the larger organism. This process of differentiation is con-trolled, at least in large part, by the action and organization of the original genome. We propose here an idealized mechanism for developing what we call composite genomes, i.e., genomes capable producing differentiated offspring. Their formation is desir-able since they provide a means for coordinating the actions of independent, functionally interacting units operating in a common environment By replicating itself many times and placing the offspring in appropriate environments, a single composite genome produces a system of implicitly coordi-nated independent units capable of pursuing a common objective. Furthermore, all of the information about the sys-tem is located in a single unit, (of which there are many copies), making communication of the group strategy sim-ple. Composite genomes are in large part responsible for the structure of the recursive, holonic hierarchies [3] which are characteristic of living systems. Our basic hypothesis is that composite genomes arise from the consolidation of specialized genetic programs which

control independent units that produce behaviors which are symbiotically related. Via a reproductive opera-tion we call hybridization, genetic programs which produce distinct specialized behaviors are combined into a single genome capable of producing either behavior. Which of the encoded behaviors replicated offspring of a composite genome exhibit is determined by the internal and external environmental conditions in which they are placed. The Basic Model In our idealized model of the natural genetic system, an environment is defined in terms of a space, which we visual-ize as a two-dimensional integer grid of infinite extent, the standard setting for cellular automata. Within this grid, environmental conditions are defined in a local way, i.e., conditions exist in regions of the space and vary with time and/or the action of the living systems in the region. Living systems are modeled as abstract genomes which reside at some location in the grid and interact with the conditions present in some symmetric region surrounding them. Each genome is capable of: 1) detecting the conditions present in its immediate external environment; 2) detecting the conditions present in its internal environment; 3) reacting to sets of external and internal states, either by establishing some internal condition, or by initiating a process capable of interacting with, and possibly altering, the conditions pre-sent in its immediate environment. Also associated with each genome in our model is a num-ber, denoting its strength. A genome's strength is initial-ly assigned by the environment and is adjusted at period-ic intervals to reflect the fitness of the genome with respect to it. It is intended as an analog to the natural sys-tem, in that it provides the basis for selection. When a genome's strength falls below a certain threshold, it "dies", and is removed from the grid, but whenever its strength 162

climbs above a somewhat higher threshold, it is able to pro-

duce an offspring. In this way, the number of offspring

allocated to individuals is biased by performance. The

initial strength of an offspring comes from its parents,

depleting their strengths and thus restricting the frequency

of individual reproduction. The principal genetic operators in

the model are crossover and replication. (Augmenting these

are idealized mutation and inversion operators. To simplify the presentation, their action is not discussed here; we utilize them in the standard fashion [1,4,5]).

The environmental grid mediates functional interaction between genomes. The potential for functional interaction exists wherever the processes initiated by one genome can, through the medium of a common environment, establish conditions which affect, either positively or negatively, the strength of another genome. In our model, genomes which are proximate to one another on the grid experience similar environmental conditions and can simultaneously affect their mutual environment Since a genome's fitness is a function of its performance within its immediate environment, the fitnesses of genomes which share a common region of the grid are linked.

The grid also performs an important role in mediating reproductive interactions between genomes. In order to more accurately model the natural system, mate selection and offspring placement are spatially biased. The probability of an offspring emerging from a crossover between two parent genomes is inversely related to the distance between them. Similarly, when an offspring is placed in the grid, the probability that that offspring is found a distance

x from its parents decreases inversely with the magnitude

of x. These spatial dependencies appear to us to be important factors in promoting the formation of composite genomes, as we discuss below.

Structural and Spatial Coherence

In distributed populations, two kinds of groups emerge due to the action of the genetic operators and regional variations in environmental conditions. These groups derive their coherence from different properties and each serves a different function. The first of these are structurally coherent groups, or genotypes. The members of a genotype can be said to be structurally coherent in the sense that the structures of their programs, and hence the behaviors these programs produce, are similar. In sufficiently large, unevenly distributed populations, we expect a number of diverse genotypes to emerge due to functional specialization.

If the individual genomes in an unevenly distributed population are computationally limited, in the sense that they are incapable of responding, simultaneously, to all the demands and regularities present in their environment, then functional specialization will occur [6]. Spatially distributed genomes under selection pressure and the action of genetic operators, will begin to pursue different

survival strategies and, after a time, will be separable

into distinct genotypes according to their patterns of behav-ior and the structure of the programs which encode for this behavior. As time goes on, the behaviors encoded in these specialized genotypes will become mutu-ally exclusive, in the sense that a given genome, due to its limited computational capabilities, will be unable to ade-quately perform more than one of the behaviors at a time. The second kind of grouping arises as a consequence of the spatial dependencies built into our model. Under our formulation, a collection of genomes which occupies a par-ticular region forms a spatially coherent group, or sub-popu-lation. These spatial groups hold a special place in dis-tributed systems. Not only do the individuals within them share a common context, but since mate selection and offspring placement are spatially biased, reproductive activity tends to be concentrated within them. If the population density is sufficiently low, individuals will tend to cluster in these spatially coherent groups, particu-larly if certain regions of the environment are more "hospitable" than others. The individuals within such "spatial niches" [7] will form isolated sub-populations whose members interact both functionally and reproduc-tively, and tend to place their offspring back in the niche with high probability. Since the members of these groups interact functionally, if the niche contains genomes from dif-ferent genotypes, the potential for symbiosis between indi-viduals from these different types exists and is selected for. A Reproductive Continuum Crossovers of parents within and between the groups described above produce offspring of different characters, each of which perform a different function in the distribut-ed system. The reproductive operations induced by the crossover operator take on the character of a continuum, pro-ducing different kinds of offspring depending on the degree of structural similarity between the crossing par-ents. This continuum is depicted below, linking two natu-rally occurring operations, replication and recombination with a third operator which we refer to as hybridization. Admittedly, it may seem odd at iirst to assert that any kind of continuum exists between replication and recombi-nation, since so many differences exist between them. Replication, the process by which a single composite genome produces a macro-organism composed of millions of differentiated cells, seems to have little to do with recombination, the process which produces new indi-viduals by "mixing" two parent genomes. They occur at different levels in the biological hierarchy and the physical process which implement them are quite distinct

Never-theless, we argue that, from an information processing per-spective, replication and recombination can be consid-ered as points on a logical continuum of reproductive operations by which new genomes can be produced from 163 Replication Recombination Hybridization D ecre«.sing Similarity existing ones. While only certain regions of this continuum are realized in the natural genetic system, we have incorpo-rated all of them in our model. Replication lies at one extreme of this hypothetical con-tinuum, occuring in our model either from the action of the replication operator or as a crossover between identi-cal parents. The resulting offspring is an exact replica of the parental genome, but, as with all genomes in our model, its actions and responses are dictated by the state of its internal and external environment. Replica-tion is most interesting when the parental genome is com-posite. When this is the case, the sections of code which become active in the offspring may differ from those which are active in the parent, due to differences in their internal and external environments. For example, a composite genome containing code for two distinct pro-cesses, each one sensitive to a distinct set of internal and external cues, can produce two differentiated types of offspring. While each is an exact genetic replica of the par-ent, different components of the composite genome are active in each of the offspring types, due to differences in the internal and external environments in which they have been placed. As a result the two types exhibit different behaviors. When crossovers occur between genotypic variants, i.e., non-identical members of the same genotype, the off-spring genome begins to look less like a replica of its par-ents and more like a mix of two similar, but distinct par-ents. In place of replication, we get recombination. Those familiar with genetic algorithms will recognize recombina-tion as the standard image of crossover. Two parents, variants of the same basic genotype, cross to produce an offspring which shares sections of both parents' genetic code, and exhibits behaviors characteristic of both par-ents, as well as new behaviors arising from epistatic inter-actions between the spliced sections. Recombination has been studied extensively by a number of genetic algortihm researchers [1,4,5], and has been shown, under certain con-ditions, to generate individuals of increasing fitness through an intrinsically parallel search of potential genospace. As the parents become less and less similar and the behaviors they encode for increasingly specialized, crossovers in our model tend toward hybridizations. We define hybridization as a crossover between genomes from radically different

genotypes, each of which codes for a separate process activated by a distinct set of internal and external cues, which produces a new, composite genome that contains the code for both processes. It is the primary source of composite genomes in our model. In hybridization, two parent genomes, each of which codes for a different, specialized process, cross at a non-inter-fering point to form a composite offspring capable of exhibit-ing either behavior, depending on the external and internal conditions it encounters. Composite Genomes We see evolution in the distributed genetic system as an interaction of the operations described above. As new indi-vidual strategies emerge, recombinations between genomes which follow similar strategies produce increasingly fit indi-viduals. When the individuals are computationally limited and spatially distributed, we expect a number of mutually exclusive strategies to emerge. If migrations occur in the space, we expect the formation of spatially coherent groups that contain genomes from more than one of these special-ized genotypes. The key to the development of successful composite genomes lies in hybridizations that occur within these structurally diverse sub-populations between symbiotically related parents. When a spatially coherent group, com-posed of genomes from more than one genotype, each of which exhibits a different specialized behavior, per-sists over time, spatial biases in mate selection and off-spring placement increase the frequency of hybrid off-spring, generated by parents from different genotypes within the spatial niche. Those offspring which successful-ly hybridize mutually beneficial behaviors will gain selective advantage. We suggest that hybridizations which occur between the genomes in spatially coherent groups whose members are symbiotically related give rise to composite genotypes which make structurally explicit the implicit, spatial coherence that exists between the members of such groups. Replications of these composite types produce a set of independent computational units which act as an implicitly coordinated, distributed system. Preliminary verification of our hypothesis that suc-cessful composite genomes can form from the spatially-biased interaction of computationally limited adaptive units was provided by an experiment with the authors' software system, Asgard. Asgard is designed to simulate the behavior and adaptation of artificial animals operating under a genetic algorithm; similar systems have also been developed by Holland and Reitman [8], Booker [9], and Wilson [10]. Unlike these systems, however, the main objective of Asgard was the study of the evolution of a distributed population of interacting genomes, rather

than the evolution of functionally isolated individuals. In the next section we describe the organization of the system and review the results of the most interesting simulation.
ASGARD Asgard's environment is a finite, toroidal, two-dimen-sional grid, with a resolution of 1 6 0 × 6 0 , which contains "food" at various locations. It is divided into 4 equal quad-rants, each of which can be displayed on a Tektronix 4105 graphics terminal. The display style is similar to Wilson's [10]. The grid is "home" to a population of genomes which move about the grid in search of food. Each time step, the genomes in the grid expend one unit of strength in order to stay "alive", and each unit of food they consume increases their strength by one unit. In order for a genome to survive, then, its average food consumption must be at least one unit per time step; in order to reproduce, its average con-sumption must be somewhat greater than 1. The behavior of each genome is controlled by its individual program. These programs are lists of labeled instructions of 164

two types, Move and Food?, which specify either a

movement direction or a test and transfer of control. Order

of execution is controlled by a program counter that speci-

fies which instruction in the list is to be performed next.

Move instructions take one argument, which specifies one

of eight directions of movement (N, E, SW, etc.). When exe-

cuted, these instructions move the genome one unit in the

specified direction. Food? instructions transfer control by

testing the eight locations surrounding the genome for

the presence of food. They take two arguments, both

labels, which specify the statement to which the program

counter should point given that food is or is not present in

any of the surrounding locations. A short example is given

below, together with a potential pattern of movement.
replicate a to create a'; begin { with probability P c }
: choose m at random from the set A (t) - {a}, biasing choice by distance |Loc(a) - Loc(m)|; crossover a' and m;

replace a'with one of results of crossover; end choose Loc(a') at random, biasing the choice by the distance |Loc(a')-Loc(a)|; coend cobegin { for all a in A(t) such that Str (a) < 1 } : delete a from A(t); coend Increment t; where: P = probability of crossover; T r = reproduction threshold; Str (a) = strength of genome a at time t; Food (Loc(a)) = number of food units present at a 's current grid location. The reader will note that Asgard'% algorithm differs in some respects from more standard implementations, (see [1,44]). For example, the population size in Asgard is variable, governed only by the carrying capacity of the environment Also, mate selection and offspring placement are biased by distance as our model specifies. These modifi-cations were undertaken in order to make Asgard more closely resemble a community and allow for the formation of semi-isolated sub-populations. Within these sub-populations, trials are allocated to individuals and schemata in proportion to their observed fitnesses, insuring that Holland intrinsic parallelism theorems hold [1]. The Task The task set before Asgard'% population is to identify and exploit the regularities present in the pattern of food place-ment. Although we performed experiments with a num-ber of patterns, of various complexities, we discuss only one here, the so-called "seasonal" pattern. Under this pattern, there are three basic regularities to which the population can adapt. 1) Food can appear in only 1/8 of the space, concen-trated in eight evenly-spaced fertile regions. Each genome must periodically visit one of these fertile regions in order to survive. 2) The productive capacity of each fertile region oscil-lates periodically between a maximum and a mini-mum value, (hence the term "seasonal"). Each fertile area can support many more genomes dur-ing its "summer" than its "winter", so competi-tion for food within a particular area becomes increasingly fierce as winter approaches. start (no) (yes) •> etc. (no) ?(no) ? ? (yes) (yes) 1 Move S Food? 2 2 Move E Food? 2 3 Move W Food? 2 1 3 3

In terms of our model, each genome's immediate external

environment is defined solely by the concentrations of food

in its neighborhood. A genome can modify its external

environment in two ways, either by consuming food or by

moving to a different location in the grid. A genome's

internal state is defined by the position of its program

counter. By executing Food? instructions, the genome can detect the state of its external environment and, based on this information, can change its internal state, potentially altering its future external response pattern.

The evolution of the population is driven by an algorithm from the class of reproductive plans [1]. The algorithm starts with a population of genomes whose programs are formed at random from the set of legal instructions. Food is placed into the environment according to a pattern which may depend on time as well as the behavior of genomes in regions of the torus. The objective is to produce a population which responds to regularities in this pattern. The algorithm produces a sequence of populations whose members interact with each other through common local environments and are allocated offspring on the basis of their individual consumption. Over time, the successive populations will contain individuals adapted not only to the conditions of the artificial environment, but also to the behavior patterns of the other members of the population.

Let A(t) denote the population at time t and let a be any member of A(t). One iteration of the algorithm consists of the steps below:

cobegin { for all a in A(t) } :

```
determine Loc (a) based on a's program and update program
counter [allows a one Move];

consume food at Loc(a);

coend

cobegin { for all a in A(t) such that Str (a) > T f } :
```
165 3) The amount of food actually produced during a giv-en time step by a fertile region is linked to the amount of food consumed in the region during the previous time step. At any given time, an opti-mal consumption level exists for a region, and the genomes in that region can over- or under-con-sume with respect to this level, as a group, dur-ing a given time step, resulting in decreased food production in the next time step. In each of the four quadrants of the grid, two 10x15 areas are established as fertile, (see figure la). For con-venience, these areas are called farms. They are the only areas in the grid where food can be found; the rest of the grid is desert-like. Each time step, Ki(t) units of food are produced by the i'th farm and distributed randomly within it, (i = 1,,,8). Ki(t) is itself a function of two other quantities: Mi(t), the seasonal productive potential of farm i; and Ei(t), the consumption efficiency within farm i, a value derived from the total consumption within farm i during the previous period, Ci(t-l). The exact form of these rela-tionships i s : Mj(t) = A s i n ((2 n t / Y) + I I (l) i) + M 0 , A < M Q ; K i (t) = E i (t) M i (t) where Y is the length of Asgard'% year. The strength of genomes in the grid is based solely on their ability to consume; the more they consume, the stronger they become and the more offspring they have. In the absence of the factor Ei(t) then, the optimal strategy for each genome would be to consume as much food as pos-sible, at every time step. This "greedy" strategy leads to global extinction, however, when Ki(t) depends on Ei(t), since future food supplies are then tied to present rates of consumption. So the seasonal pattern exerts two differ-ent pressures on the members of the population. On the one hand, each genome is encouraged to compete with all the other genomes in the population for every available unit of food, in order to increase its individual strength. On the other hand, the environment gives selective advan-tage to groups of genomes which manage to cooperate in restricting their group consumption rate to track optimum levels. Given the limited capabilities of each genome, however, it is clear that the forces which coordinate any group actions must come from some source outside the individual. Indi-viduals are so limited in their range of
```
```

options that they can respond to only the first two regularities of the pattern, i.e., they can seek fertile farms and, on finding them, remain near them. In order to respond to the third condi-tion, group consumption levels must somehow be regu-lated, and this is simply impossible from an individual per-spective. The individual genomes have no way to communi-cate with the other members of the population, nor do they have any direct means for changing one another's behavior. In our model, the selective pressure which the third reg-ularity places on groups of individuals is the agent which produces coordinated action. The selective advantage giv-en to groups of individuals which succeed in maximizing the productive efficiency of their farm assures their contin-ued survival. Thus we expect two kinds of groups to form as a result of this pressure: spatially coherent groups of structurally similar genomes whose behaviors are nearly identical and mutually complementary and spatially coherent groups of genomes from distinct, but symbiotic genotypes, each of which performs some different, but mutually compati-ble behavior. Farmers and Nomads In studying the evolution of populations in Asgard, our objective was to detect the pattern and character of the changes which the population underwent over time. To this end we developed a set of tools which allowed us to study populations in Asgard qualitatively, as one might a colony of living bacteria, to try to categorize and account for the changing patterns of behavior and observe the evolu-tion of the community as it developed. The interactive display thus became our primary means for studying the system. As the size of our experiments increased, (typical popula-tion sizes were on the order of three to four hundred indi-viduals and required almost 48 hours of CPU time on a Prime S50 to complete), observation in real-time soon proved unwieldy. In place of real-time observation, we substituted a kind of "batch" mode, which allowed more extensive study of the population at periodic intervals. A "snapshot", which preserved the complete state of the sys-tem, was taken every 300 iterations. After every "snapshot", each of the genomes in the population was automatically classified into a genotype based on similarities in program structure. Each of the genotypes' behavior patterns was then studied separately. For each genotype, Asgard was run interactively, using a population of representative individuals from that genotype, and their behaviors noted. After each genotype had been studied in isolation, Asgard was again run interactively, this time re-starting the simulation with all the data from the "snapshot", to study the behavior of the population as a whole. (In this mode, the genomes of each genotype were displayed by a different

letter, thus making it possible to visually identify each genome's "species".) We give the results from one of these qualitative studies here. An ini-tial set of 1000 individuals was constructed at random, their strengths set to SO, (see figure lb). This initial set was entered into the grid in random locations at a gradu-ated rate, to smooth the transient behavior of the popula-tion curve. As the simulation begins, the programs of the initial 166 E i (t+i) | 2 C i Ct)-M i (t) M i (t) for 0 < C i (t) < M i (t); otherwise; , 0 1

genomes interact with the environment to produce what look like random walks. Those which have been initially placed far from any of the farms die quickly, many without ever consuming a single unit of food. In this initial stage, selection favors those genomes with the ability to recog-nize and remain in the vicinity of a fertile zone. Farm efficiencies, as measured by the value Ei(t), all average below .3. After 150 iterations the initial population col-lapse stabilizes, and the population begins to oscillate between two and three hundred individuals, evenly dis-tributed between the four quadrants.

After 1500 iterations, (see figure 1c), the population consists of genomes which can be separated by pro-gram structure into eight different genotypes, with some slight variation in each type. Some of these genotypes are present on only one farm, while others have spread, by chance migrations and reproductions, to several of the farms in the grid. All the genomes exhibit the same basic pattern of behavior at this stage. Their programs consist of various series of Move instructions which cause them

to drift in a particular direction, punctuated liberally with

Food? tests. Positive values for these tests tend to halt the drift, so that the genomes wander about in various directions and, upon encountering a fertile farm, move randomly within it Occasionally an individual will drift from one fertile farm to another, but most often those who leave a farm end up dying in the desert. Globally, the only populated areas are me regions immediately surrounding the fertile farms, the genomes there forming eight spatially coherent groups. The size of each group follows the productive capacity of the farm, climbing during the "summer", falling in "winter". In independent trials of the various

genotypes, no evidence of symbiosis was found to exist between the groups. Each of the genotypes performed as well in the absence of the other groups as they did in their presence, provided population densities were maintained at approximately the same levels. Consumption efficiencies do not exceed .4, on any farm, during any season. After 3,000 iterations however, both structurally and spatially coherent groups are present and exhibiting co-operative behavior, (see figure 1d). The two upper quad-rants are completely extinct and the farms there can no longer produce food, but the lower two quadrants now sustain an average of 350 individuals, roughly the same size population as the four quadrants once maintained. Only two significant genotypes exist, each exhibiting markedly different behavior. The first group, nicknamed the Farmers, are the descendants of the genotypes present at 1,500 iterations. A group of Farmers is present at each remaining fertile farm. As individuals, they follow a fixed, circular pattern of movement, so mat when placed in the neighborhood of a farm, following this pattern assures them of encountering it repeatedly. When tested in isola-tion, a population consisting solely of Farmers was stable and able to maintain the productive capacity of each farm in the grid. This "all Farmer" population

functioned efficient-ly as a group, maintaining average consumption efficien-cies of approximately .9 during the winter. During the sum-mer however, the Farmers did not multiply sufficiendy to keep pace with the farm's increased productive capacity and, as a result, their efficiency rating declined to around .5. The second group, nicknamed the Nomads, are a new genotype, which exhibits a kind of migratory behavior. Each Nomad moves steadily across the screen, from left to right, in a tight, zig-zag pattern. Their rate of advance is (b) Figure 1 167 (d) (c) (a) c u H V S V E B A C I Y L K G G K Z P Y D A M O w P I B T J F F 1 F F F F F F F E R F F F NN NN NN NN Winter Summer M D R R D M D E M P E D M E D M E R R Q X u N N N N N N N N synchronized to the change of seasons, so that they strike the left edge of a farm during the farm's "spring" and move across it during its most fertile period, leaving the right-most edge of the farm as its productive potential wanes. The Nomads are found in two separate groups, which are evenly spaced between the farms in the lower two quad-rants. Each group is largest as it leaves the right-most edge of a farm. As me group crosses the "desert" between farms, its numbers decrease steadily until the next farm is reached. Unlike the Farmers, when a population of Nomads was tested in isolation, they could not survive. In following their pattern of cyclic migration, the Nomads failed to main-tain consumption levels on farms during tiieir "winter" sea-son. As a result, only a fraction of the food available the first summer was available in the summer of the second year. Without Fanners to build up food stocks during the winter and early spring, the amount of food available to arriving groups of Nomads was gradually reduced. This decline in the summer food supply began a vicious cycle of declining numbers which ended in the extinction of the test group. When observed together, however, the two structural-ly coherent groups complemented each other's efforts. With both Farmers and Nomads present in the same pop-ulation, consumption efficiencies were at all times greater than .8. Recall mat, operating in isolation, groups of Farm-ers were able to maintain near optimal consumption rates during the winter, but that during the summer their efficien-cy fell significandy. When Nomads and Farmers were observed together, winter consumption rates were main-tained by the groups of Farmers at approximately the same level as when the Farmers were tested in isolation. (Dur-ing the winter the two groups of Nomads were crossing the desert, and thus did not affect consumption rates.) When a Nomad group arrived at a farm during its "spring" though, the population influx

increased the consumption rate within the farm enough to bring the value closer to the optimum. As a result, summer consumption efficiencies increased from .5 to .8. Both structurally and spatially coherent group behav-iors are present at this stage, then. The Farmers and the Nomads are both examples of structurally coherent, (or genotypic), groups. The implicidy coordinated actions of tiieir individual members, acting on a local scale, (i.e., pursuing particular patterns of movement), add up to a group action on a global scale, i.e., the regulation of group consumption rates. And on each farm, during its summer, members of both genotypes, Fanners and Nomads, form spatially coherent groups which act together to increase farm efficiencies during mat season. These increases are beneficial to both genotypes, since the increased food production allows both groups to generate more offspring. Because of spatially biased mate selection, many of these offspring are crosses between Nomad and Fanner. Functional specialization is present in the population at this stage however, and as a result, little in the way of improvement can be expected from recombining Farmers and Nomads. Farmer behavior is sufficiendy different from Nomad behavior that an offspring which shares some of the traits of each group more often than not does a poor job at both. The Farmer and Nomad function well together, but only as separate, independent units. Each genome is limited enough in complexity that it cannot satisfactorily perform both functions. What is required is a hybridization combining the two behaviors into a single genome capable of producing both types of offspring. A successful composite genome emerges after 4,000 iter-ations which consolidates the symbiotic relationship between Fanner and Nomad into a single program. At the top of this Farmer/Nomad composite is a section of code which acts as a kind of "switch". This switch is actually a series of linked Food? tests. When an offspring of this composite genome is placed in the grid, it tests several times for the presence of food. If two in a row of these tests turn up positive, the offspring's program counter moves to a set of statements which correspond to an infinite loop which codes for Farmer behavior; if two in a row turn up negative, the counter moves to a different set of state-ments, and the offspring performs an infinite loop which causes it to act like a Nomad. The population composed entirely of composite forms per-formed almost exactly as the population we described at 3,000 iterations. The only difference is mat the implicit link between the two groups, Farmer and Nomad, is now explicitly expressed in a single piece of code. A single copy of the composite genome, placed in fertile conditions

and allowed to replicate, produces offspring which spread throughout the space and spontaneously organize them-selves into the symbiotic groups of Farmers and Nomads which sustain the farms in the environment at near optimal levels. DISCUSSION The results of the Asgard experiments are encouraging. The emergence of the composite Farmer/Nomad shows that it is possible, through hybridization, to evolve a composite genome capable of producing and organizing a distributed system of independent units which work toward a common goal. Asgard provides another example of the versatility and adaptive power which genetic algorithms can exhibit, and as a prototypical example of the use of a genetic procedure in a distributed context, it gives insight into which facets of the natural system may be important in forming coordinated, co-adapted systems of distributed processes. However, work witii Asgard was hampered by a number of characteristics which made experimentation with the system unwieldy. For example, Asgard's environ-ments were extremely sensitive to slight changes in a number of internal parameters, and unforeseen interactions between environmental components often led to extinc-tions or population explosions. Attempts to extend Asgard to more useful domains proved to be exercises in frustra-tion. Furthermore, we decided that in order to imple-ment the distributed genetic algorithm in more realistic 168

domains, some substitute source for the contextual

clues provided by the environmental space would have to

be developed.

To address some of these questions, we are currently

developing a new system, Midgard, designed to produce

an adaptive solution to the problem of balancing the load in

a distributed interconnected local area network (inter-

LAN). If communication costs between the nodes on an

inter-LAN are sufficiently small, it is possible to achieve

gains in overall efficiency by reassigning jobs between

nodes in the network to exploit under utilized computing

resources [11]. The algorithms which control this load

balancing process are themselves typically distributed. Each node runs a scheduling process, or job scheduler, which communicates with the other schedulers in the network to determine when and how to migrate jobs between processors. Midgard is designed to explore the utility of a distributed, genetically based, adaptive algorithm in developing co-adapted process schedulers capable of cooperating to improve network performance. The networks we consider are inter-LANs, in the class of systems which Fuller and Siewiorek categorize as loosely coupled [12]. The environmental grid, which in our model implicitly groups genomes operating in common contexts, is replaced in Midgard by explicit classes which are induced from the hierarchical structure of the network and the patterns of functional interaction which arise between its nodes. Distance is replaced by class membership as the measure of contextual proximity. Structurally coherent groups are encouraged by biasing mate selection and offspring placement to occur within a LAN cluster or processor class. Encouraging reproductive interaction within these groups fosters the formation of genotypes in the system organized according to the symmetries which exist between processors in the network.[[7]] By increasing the number of crossovers between similar machines, recombinant offspring will be

generated which gradually increase the performance of the machines within clusters and within processor classes. An analog to the spatially coherent group emerges when mate selection and offspring placement are biased to occur more often within classes induced by the occurrence of successful migrations between machines of different classes. By monitoring the patterns of successful migrations over time, reproductive interaction can be concentrated between individuals, from different genotypes, which have successfully co-operated to increase network efficiency. Favoring matings within these groups allows subpopulations working on common goals to become semi-isolated reproductively, setting the stage for successful hybridizations which explicitly capture group interdependencies.

The software to implement Midgard is currently under development in C on a unix-based Vax 8600. The network which the schedulers drive is implemented using C-SIM, a network simulation package developed by Schwebtman at Purdue University. Results from the

it only needs to know which classifiers directly activated it and which classifiers it directly activates. There is no need for complicated book-keeping or for high-level critics to analyze sequences of actions and assign credit accordingly. Second, the bucket brigade works in a highly parallel way, changing the strength of many (or all) rules at the same time. Third, the bucket brigade acts incrementally, changing the strength of classifiers gradually. By changing the strength of classifiers only a small amount at a time, the classifier system tends to learn gracefully, without the precipitous changes in performance that may result from making a large change in response to a single, possibly anomolous, case.

One key issue for systems using the bucket brigade algorithm is how fast strength flows down long sequences of classifiers. If a whole sequence of classifiers must be activated many times in order to adjust the strength of a classifier at the beginning of the chain in response to a change in payoff associated with the last step in the sequence, the system's response to simple changes in its environment will be too slow. Wilson [Wilson, 1986] used a simple simulation to show that allocation down a sequence of classifiers

can take a fairly large number of steps. (He suggested an alternative "hierarchical" bucket brigade algorithm that is designed to speed up the flow of credit down long sequences of classifiers.) Holland [Holland,1985] mentions this problem and suggests a way to implement "bridging" classifiers that speed up the flow of strength down a long sequence of classifiers.

This paper describes some simple experiments designed to show how well the bucket brigade is .able to allocate strength down long sequences of classifiers. Section 2 describes the CFS-C/FSWl classifier system, which is used to carry out all experiments described in this paper. In Section 3, the allocation of strength down a single chain of classifiers will be examined. In Sections 4 and 5, the ability of the bucket brigade to allocate strength so that the system learns to make the proper choice at the beginning of a long sequence of steps is examined. The effects of "bridging" classifiers are also examined. In Section 6 the effect of sharing classifiers in different sequences is examined, without and with bridging classifiers.

2 THE C F S C / F S W l SYSTEM

All experiments described in this paper were done using the CFS-C classifier system [Riolo, 1986], set in the FSW1 ("Finite State World 1") task environment [Riolo, 1987).

This section describes the parts of the CFS-C/FSWl sys-

tem that are relevant to the experiments described in this

paper. For a complete description of those systems, see the

documentation cited. Basically, the FSW1 domain is a world that is modeled

as a finite Markov process, in which a payoff is associated

with some states. The classifier system's input interface provides a message that indicates the current state of the Markov process. The classifier's output interface provides the system with a way to alter the transition probabilities of the process, so that the system can control (in part) the path taken through the finite state world. When the clas-sifier system visits states with non-zero payoff, that payoff is given to the system as a "reward". Thus the task for the CFS-C classifier system in the FSW1 domain is to learn to emit the appropriate signals at each step so that the Marlov process will visit states with higher payoff values as often as possible. More formally, the FSW1 task domain is fully defined by specifying: • A set of n states W i , i = 0, ..., n - 1, each with an associated payoff μ(W i) € SR; one state also is desig-nated the start state. • A set of probability transition matrices, P(r), where each entry Pij(r) in P(r) gives the probability of going to state W j , given that the system is in state W i and that the classifier system has emitted r as its output value (r = 0...15). Figure 1: A simple FSW1 finite state world. For example, consider the simple three state world shown in Figure 1. (In this and other diagrams, states are shown as circles, and arrows designate non-zero prob-ability transitions.) W 0 is the start state. The payoff for state W 1 is 100; the payoff for the other states is 0. When the system is in state W 0 , if r = 1 the probability of going from state W 0 to W 1 is 1.0; if r = 2 the probability of go-ing from state W 0 to W 2 , is 1.0. For other values of r, the probability of going from state W 0 to state W 1 or W 2 is 0. The probability of going from either W 1 or W 2 to W 0 is 1.0, no matter what the value of r. Thus if the classifier system is to maximize its payoff in this world, it must learn to set r = 1 whenever it is in state W 0 . The CFS-C classifier system is a standard, "Holland" type learning classifier system that consists of four basic parts: • A message list, which acts as a "blackboard" for com-munications and short term memory. In the CFS-C

classifier system, the message list has a small, maxi-mum size. 185 W 1 r=1 μ=100 W 0 r=2 W 2 • A classifier list, which consists of condition-action rules called classifiers. Each classifier in the CFS-C system is a two-condition classifier of the form: C 1 ,C 2 /Action A classifier's condition part is satisfied when each of the conditions C 1 and C 2 is matched by one or more messages on the message list. The second con dition may be prefixed by a "~", in which case that condition is satisfied only when no message matches the condition C 2 . A satisfied classifier produces one message for each message that matches its first con dition, C 1 ,using the usual "pass through" procedure. Each classifier also has an associated strength, which is related to its usefulness in attaining rewards for the system, and a specificity (sometimes called its bid ratio), which is a measure of the generality of the classifier's conditions. • An input interface, which provides the classifier sys tem with information about its environment. In the FSW1 domain, the input interface provides one de tector message which indicates the current state of the Markov process. • An output interface, which provides a way for the classifier system to communicate with or change its environment. In the FSWl domain, the output in terface maps messages that start with a "10" (some times called effector messages) into an effector set ting, r, r = 0 . . . 15, which determines the transition probability matrix P(r) used to select the next state of the Markov process. As in other "Holland" classifier systems, messages are all strings of fixed length l, built from the alphabet {0,1}. Each condition C i and the action part of a classifier is also a string of length l, built from the alphabet {0,1, # } . The # acts as a "wildcard" symbol in the condition strings, and it acts as the "pass-through" symbol in the action part of a classifier. The CFS-C/FSW1 system is run by repeatedly execut ing the following steps of the classifier system's "major cy cle": 1. Add messages generated by the input interface to the message list. In the FSWl domain one message, which indicates the current state W(t) of the world, is added to the message list. 2. Compare all messages to all conditions of all classi fiers and record all matches for classifiers that have their condition parts satisfied. 3. Generate new messsges by activating satisfied classi fiers. If activating all the satisfied classifiers would produce more messages than will fit on the message list, a competition is run to determine which classi fiers are to be activated. Classifiers are chosen prob abilistically, without replacement, until the message list is full. The probability that a given classifier is activated is

proportional to its bid. 4. Process the new messages through the output inter face, resolving conflicts and selecting one effector set ting, r, to be used for the current time step. Once r is set, the associated transition matrix P(r) and the current state W(t) are used to select the world state W(t + 1) to which the system moves. 5. Apply the bucket brigade algorithm, to redistribute strength from the environment to the system and from classifiers to other classifiers. 6. Apply discovery algorithms, to create new classifiers and remove classifiers that have not been useful. 7. Replace the contents of the message list with the new messages, and return to step 1. In the CFS-C/FSW1 system, the bid of classifier i at step r, B i (t), is calculated as follows 1 : B i (t) = k∗ S i (t) ∗ BidRatio i k is a small constant (usually about 0.1), which acts as a "risk factor", i.e., it determines what proportion of a clas sifier's strength it will bid and so perhaps lose on a single step. 5,(1) is the classifier's strength at step t. BidRatio i is a number between 0 and 1 that is a measure of the clas sifier's specificity, i.e., how many different messages it can match. A BidRatio of 1 means the classifier matches ex actly one message, while a BidRatio of 0 means the clas sifier matches all messages. When a competition is run to determine which classi fiers are to be activated, the probability that a given (sat isfied) classifier i will win is: 1 Actually the CFS-C/FSWl bid calculation involves other param eters not shown here, but for the experiments described in this paper, those parameters have been disabled. 186 Prob(i wins) = β i (t)/ Σ i β i ,(t) β i (t)=B i (t) BidPow The effective bid, β i (t) , of a classifier i at step t, is: BidPow is a parameter that can be set to alter the shape of the probability distribution used to choose classifiers to produce messages. For example, if BidPow = 1 then β i (t) = Bi(t), i.e., a classifier's probability of producing messages is just its bid divided by the sum of bids made by all satisfied classifiers. Setting BidPow to 2, 3, and so on, makes it more likely that classifiers with the highest bids

will win the competition. The effects of varying BidPow

are considered further in section 4 of this paper.

Note that the output interface of the CFS-C/FSWl sys

tem may have to resolve conflicts, e.g., when one classifier

produces a message that says "set the effector value r to 1"

and another produces a message that says "set the effector

value r to 2". Since the effector can only be set to one value

at a time (just as we can either lift our arm or lower it, and

not both), an effector conflict resolution mechanism must

be used. Basically, when there are conflicts the value of r

is chosen probabilistically, with the probability that r = r' where S,(t) is the strength of classifier t at t, R(t) is the re ward from the environment during step (t), Pi(t) is the sum of all payments to classifier » from classifiers that matched messages produced by t during the previous step, and B,-(t) is the classifier's bid during step t. Clearly a classifier's strength reaches a fixed-point when the amount of strength it receives is equal to the amount it pays. Thus in the long run a classifier's fixed-point strength, S fB , approaches:
Figure 2: A single path of states leading to a reward at state W 12 . When the system reaches state W 12 , it will go to state W 0 by default; i.e., p ij (0) = 1.0 for i = 12,j = 0. (In this and subsequent descriptions of finite state worlds, all transitions not mentioned have probability 0.) Thus with a perfect set of classifiers, the system could achieve a reward of 100 every 13 cycles. The CFS-C/FSWl system was run in this world with twelve classifiers, each with a starting strength of 50. Clas sifier 1 was of the form: m' S i (t +1) = S i (t) + R(t) + P i (t) - B i (t)

where m t ranges over the effector messages that say "set r

to r'", β m t (t) is the effective bid of the classifier that posted

message m t , m ranges over all effector messages, and β m (t)

is the effective bid of the classifier that posted message m.

The winning r value is used to select a transition matrix

P(r), which in turn is used to determine to which state

the system will move. Once an effector value is chosen, all

messages that are inconsistent with that setting are deleted

from the new message list.

The basis for the reallocation of strength done by the

bucket brigade algorithm is payoffs received by the classifier

system from the FSWl environment. In the CFS-C/FSWl

system, when the Markov process enters a state W i all clas

sifiers that posted messages which are on the new message

list (after any effector conflicts are resolved) have the full

payoff μ(W i) added to their strength. Thus when the acti

vation of a classifier tends to be directly associated with a

high reward from the environment, that classifier's strength

is on average increased. The bucket brigade algorithm also redistributes strength

from classifiers to other classifiers. In particular, when a

classifier posts messages, it pays the amount it bid to the

classifiers that made it possible for that classifier to be

come active. Let BidShare equal the classifier's bid, B,

divided by m, number of messages that matched its condi

tions. Then the strength of the active classifier is decreased

by BidShare*m and BidShare is added to the strength of

each classifier that produced a message that matched the

activated classifier's conditions. (If a classifier is

matched

by one or more "detector" messages, i.e., messages from

the system's input interface, the classifier's strength is still

decremented by BidShare for each detector message used,

but that amount is not added to the strength of any other

classifier. Thus just as the environment is the ultimate

source of strength, in the form of payoffs, it is also the ulti

mate sink for strength, when detector messages are used.) In summary, the strength S i (t + 1) of a classifier i at t+1 is: d 0 ,d 0 /e 1 r = l where each condition "d 0 " matches the detector message for state W 0 , and the action "e 1 , r = 1" posts an effector mes sage e 1 that sets the effector value r to 1. (In order to r=1 r=1 r=1 r=1 μ = 100 W 12 W 11 W 1 W 0 S fp = (R + P)/(k∗ BidRatio) where R and P are the average amounts the classifier re ceives per activation as rewards from the environment and payments from other classifiers, respectively. Since the focus of this paper is on the allocation of strength by the bucket brigade algorithm amoung existing classifiers, rather than the creation of new classifiers, the CFS-C/FSWl system's rule discovery algorithms are not used in the experiments described in this paper. Instead, all classifiers are added to the initial classifier list. Those clas sifiers remain unchanged, except for their strengths, during the course of each experiment. 3 SIMPLE SEQUENCES OF CLASSIFIERS To get a feel for how the bucket brigade algorithm works in the FSWl domain, consider the finite state world shown in Figure 2. There are 13 states in this world, W i , i = 0, . . . , 12. State W 12 has an associated payoff of 100, while the payoff for all other states is 0. The start state is W 0 . The classifier system must set r = 1 to move from state W 0 to W 12 ; i.e., p i j (l) = 1.0 for j = i + 1,i = 0 . . .11 . 187 m Σβ m' (t)/Σβ m (t) where the condition "e i -1" matches the effector message produced by classifier i − 1, i = 2, . . . , 12, and the ac-tion part of each classifier i produces an effector message e i which sets r to 1. In short, when the Markov process enters state W 0 , clas-sifier 1 is activated, which posts an effector message that moves the system to state W 1 . Since classifier 1 is activated by detector

messages, it pays its bid to the system rather than to some other classifier. At the next time step, clas-sifier 2 is activated by the message produced by classifier 1, so classifier 2 pays its entire bid to classifier 1. Classifier 2 also posts an effector message that moves the system to state W 2 . Thus the process continues, each classifier be-ing activated by and paying its bid to the classifier that was active on the prior step. Finally the Markov process reaches state W 12 in which case the classifier active during that step, classifier 12, receives the payoff associated with that state (100). The system then returns to state W 0 , and the cycle starts again. Figure 3 shows the results of running the 12 classifiers described above for a period of 3000 cycles (about 230 passes through the sequence), using k = 0.1 and BidPow = 1. The strength of classifiers 1, 4, 7, 10, 11, and 12 are shown plotted against the number of time steps executed. Figure 3 clearly shows the wave of strength flowing from Figure 3: The flow of strength down a simple sequence of coupled classifiers. Classifier 12 leads directly to a reward, and classifier 1 is at the start of the sequence. Figure 4: The number of steps required for a classifier in a chain of coupled classifiers to reach 90% of its fixed-point strength, plotted against the number of steps to the end of the chain, for k = 0.05, 0.1, 0.2, and 0.3. classifier 12, which receives the reward directly from the environment and reachs its fixed-point strength first, to classifier 1, which is farthest in the chain from the envi-ronmental reward and so reaches its fixed-point strength last. (The blips are artifacts of when strength is recorded: sometimes a classifier's strength is displayed at the end of a step in which it posted a message, so that it has just had its strength reduced by its bid but it has not yet been paid by the next classifier in the chain.) As expected, the fixed-point strength, S fp , of all the classifiers in this experiment is the same (1000), since each classifier pays its full bid to its one predecessor (or to the system for the detector message, in the case of classifier 1). The number of cycles it takes for a classifier n steps from the environmental reward to reach 90% of its S fp fits the following equation: 100C 600 S T R E N G T H 600 40C 0 600 1200 CYCLE STEP 1600 2400 3000 CLASSIFIER CLASSIFIER CLASSIFIER CLASSIFIER CLASSIFIER CLASSIFIER 1 4 7 10 11 12 DISTANCE FROM REWARD 2 5 7 10 12 0 C Y C L E S T E P S T O C R I T E R I A 1000 2000 300C 4000 BID.K = 0.05 BID.K = 0.10 BID.K = 0.20 BID.K = 0.30 t = 286 + 155n .R = 22+11.9n This number is in good agreement with the value arrived at in [Wilson, 1986], using a simple simulation of the bucket brigade. One way

to speed the flow of strength back through a sequence of classifiers is to increase k, the bid constant that specifies what proportion of a classifier's strength is to be risked on any one bid. Figure 4 shows a comparison of the 188 where n = 0 for classifier 12, n = 1 for classifier 11, and so on. In terms of the number of passes through the sequence of states (i.e., the number of rewards received), this works out to be: ei-l,e i -1/e i ,r = 1

make the classifiers more readily understandable, they will

be shown in an "interpreted" form rather than in terms

of strings built from the {0,1, # } alphabet.) Classifiers 2 through 12 are of the form: 0 200

results obtained when the 12 classifiers described earlier

were run using values of k = 0.05, 0.1, 0.2, and 0.3.

The horizontal axis shows the number of steps a classifier is

from the environmental reward. The vertical axis shows the

number of cycles it takes for a classifier to reach 90% of its

fixed-point strength. Higher values of k result in faster flow

of strength down the sequence of classifiers. For example,

for a classifier 10 steps from the reward, the number of

passes through the sequence is 59 for k = 0.3, compared to

141 for k = 0.1.

4 CHOSING BETWEEN SIMPLE SEQUENCES OF CLAS-SIFIERS

Clearly one important characteristic of the bucket brigade

algorithm is the number of cycles it takes for strength to

flow down a chain of coupled classifiers. Another important

measure of the bucket brigade algorithm's performance is

the ability of the classifier system to respond to changes in

the payoffs associated with states. For example, consider the CFS-C/FSWl world shown

in Figure 5. There are 19 states in this world. The start

state is W 0 . When the system is in W 0 , if the classifiers

set the effector value r to 1, then the system goes to state

W 1 with probability 1; if the classifiers set r to 2, then

the system goes to W 11 . Once the top or bottom path

is chosen, the Markov process can be moved through the

intervening states to W 9 or W 19 by continuing to set r to

1 or 2, respectively. When the process reaches state W 9 or

W 19 , it always returns to state W 0 . The CFS-C/FSWl system was run in the world de-

scribed above using the follwing 18 classifiers: Figure 5: Two competing paths of states leading to rewards at states W 9 and W l9 . Suppose states W 9 rnd W 19 both have a payoff of 100, and all other states have a payoff of 0. In this case it does not matter what path the system takes—the maxi-mum payoff rate it can achieve is 100 per 10 cycles. Note that the fixed-point strength of all classifiers in both se-quences is 1000 (assuming k = 0.1 and each classifier has a BidRatio = 1). In particular, classifiers 1 and 11 will have the same fixed-point strengths, so each will have a 0.50 probability of winning the competition, and so the system will go down each chain 50% of the time. Suppose the payoff assoicated with state W 9 is changed to 400. In this case the optimal payoff, 400 per 10 cy-cles, can be achieved by always taking the top path. Thus to achieve optimal performance, the bucket brigade must reallocate strength so that classifier 1, the classifier that causes the system to go down the top path, has a higher fixed-point strength than classifier 11, the one that causes the system to go down the bottom path. The faster the system can reallocate strength, the faster the system can respond to the change m its environment. Figure 6 shows the results of running the above

classi-fiers in the world descr bed above, with μ(W 9) = 400 and μ(W 19) = 100. (The results in this and the rest of the experiments described in this paper are the average of 10 runs, each started with a different seed for the system's pseudo-random number generator.) All classifiers had an initial strength of 1000, i.e., the system was started as if it had been run with W 9 = W l9 = 100 until the classifiers all reached their fixed-point strengths. BidPow was set to 1, i.e., a classifier's effective bid equals its bid. k was set to 0.1 in this and all the rest of the experiments described in this paper. Given that BidRatio = 1 for all 18 clas-sifiers, the expected fixed-point strengths for classifiers in the top and bottom chains is 4000 and 1000, respectively. The maximum payoff rate (per 200 cycles) is 8000, and the payoff expected if the choice of path is made at random is 5000. Figure 6 shows the marginal payoff the system received plotted against cycle stops executed. The strength of clas-sifiers 1 and 11, the classifiers that compete to choose the 189 μ=100 [μ=400] r=1 r=2 W 0 W 1 >r-1 w 2 ,r=1 r=1 r=1 w 8 . w 9 w 11 W 12 r=2 r=2 r=2 r=2 μ = 100 W 18 W 19 d 0, d 0 /e 1 ,r = l (1) d 0 ,d 0 /e 11 ,r = 2 (11) e i -1,e i -1/e i ,r = 1 (i = 2 . . . 9) e i -1,e i -1/e i , r = 2 (i = 1 2 . . . 19)

(The numbers in parentheses on the right serve to identity

the classifiers.) Each classifier has BidRatio = 1. Basi-

cally, the classifiers 1 and 11 compete to become active

when the system is in state W 0 . Those classifiers try to

have the system take the top or bottom path, respectively,

by (a) setting the effector value to 1 or 2, and (b) producing

a message that sets the stage for the rest of the classifiers

in its associated sequence to fire, one after another. (Note

that 1 and 2 can't post messages at the same time, since

they try to set r to different values.) For example, if classi-

fier 1 wins the competition, its message sets r to 1, so that

the system moves to state W 1 . In the next step, classifier 2 is matched by the message produced by classifier 1, so classifier 2 pays its bid to classifier 1, and the system is moved to state W 2 . This process continues until the system reaches state W 9 and classifier 9 receives the payoff associated with that state.

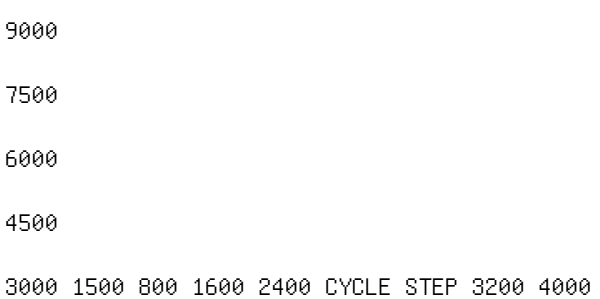


Figure 6: Marginal performance and the strength of classifiers 1 and 11 when the system is run with two competing sequences in the world shown in Figure 5. The rewards at the end of the sequences chosen by classifiers 1 and 11 were 400 and 100, respectively.

path the system takes, is also shown. Note that as the strength of classifier 1 increases toward its fixed-point value, marginal performance also increases. Let the fixed point payoff, P_{jp}, be defined as the average marginal payoff in the last one quarter of a run. Then the average P_{jp} for the runs shown in Figure 6 was 6870, which is 85.9% of the optimal payoff rate. The system's performance is less than optimal because the competition is stochastic—the higher

strength classifier has a higher probability of winning but

it doesn't always win. Also note that it takes about 1700 steps (170 trials) for

the system to reach 90% of the P jp . This is a little more

than might be expected given the results described in the

previous section; the reason for the longer observed time

is that the system often traverses the lower path, in which

case increased strength is not flowing down the top chain. One way to increase the fixed-point payoff rate is to

bias the effective bid in favor of high strength classifiers

by setting BidPow > 1. The following table compares the

results obtained for BidPow = 1,2, and 3: BidPow (%Max) 1 2 3 85.9 96.7 98.3 8000 7000 5000 3000 800 1600 2400 CYCLE STEP 3200 4000 Figure 7: Marginal performance when the system was run with competing sequences of classifiers as in Figure 6, using three different values of BidPow. As expected, increasing BidPow increases the payoff rate. Figure 7 shows the marginal performance obtained in these experiments plotted against the number of majorcycle steps executed by the system. Not only are the fixedpoint performance levels increased by increasing BidPow, but the number of steps it takes the system to respond to the change in payoff is decreased somewhat. In the rest of the experiments described in this paper, BidPow is set to 3. Even with BidPow = 3, it takes the classifier system a large number of passes down the full sequence to learn to take the path to the higher reward: as Figure 7 shows, it takes about 100-120 passes (1000-1200 cycles) to begin to respond to the change in the environment, and about 160- 170 passes (1600-1700 cycles) to reach 90% of the fixedpoint performance rate. One way suggested by Holland [Holland,1985] to speed up reallocation of strength down a long sequence of clas-sifiers is to introduce a "bridging" classifier. Basically, a bridging classifier (sometimes called an "epoch marking" or a "support" classifier) is one that is activated by a mes-sage produced by the first classifier in a sequence, and which remains active until the payoff state at the end of the sequence is reached. Since all classifiers that are active when a payoff is achieved have

that payoff added to their strengths, the bridging classifier has its strength increased the first time the sequence is executed. The next time the sequence is executed, when the bridging classifier again is activated by the message produced by the first classifier in the chain, its payment to that first classifier reflects the payoff it received on the first pass down the sequence. In 190 0 0 STRENGTH OF CLASSIFIER] STRENGTH OF CLASSIFIED MARGINAL PAYOFF TO SYSTEM 1 11 4000 M A R G I N A L P A Y O F F T O S Y S T E M 6000 BIDPOW = 1 BIDPOW = 2 BIDPOW = 3

this way, the change in payoff at the end of a long sequence

of classifiers is passed almost immediately to the classifier

at the beginning of the sequence.

To test the effectiveness of bridging classifiers, the CFS

C/FSW1 system was run using the same finite state world

shown in Figure 5, using the same 18 classifiers described

above plus two more bridging classifiers (one for the top

sequence and one for the bottom one). In particular, the

bridging classifiers were: (e 1 , | m 10), ~ d 9 /m 10 (10) (e 11 | m 20), ~ d 19 /m 20 (20)

Both of the classifiers have BidRatio = 0.5. Classifier 10,

the bridge for the top sequence, says "If the message from

classifier 1 or from classifier 10 is in the message list, and

the detector message for state W 9 is not in the list, then

post message m 10 ". Thus if classifier 1 posts a message on

step t, classifier 10 will be activated on the next step, and it

will use the message it produces to activate itself until the

system reaches state W 9 , the end of the top path.

Classifier

20 acts similarly for the bottom chain. The starting strength for all classifiers was set to the

fixed-point strength expected when the payoffs for states

W 9 and W 19 are both 100. The payoff for state W 9 was

then set to 400, and the system was run for 4000 major

cycle steps. Figure 8 compares the results obtained when the system

was run with and without the bridging classifiers. Both

marginal performance (per 200 steps) and the strength

of classifier 1 is plotted versus cycle step. (The strength

of classifier 11 doesn't change in these experiments—just

the ratio of the strength of classifier 1 to 11.) Note that

with the bridging classifier, the strength of classifier 1 be-

gins to rise almost immediately, within the first 200 cycles,

as does the marginal performance. On the other hand,

without the bridging classifier, the strength of classifier 1

and the marginal performance doesn't begin to rise until

about cycle step 1100. Similarly, with the bridging classifier

marginal performance reaches 90% of its fixed-point level

(98.4% of the maximum, StdDev = 1.1%) in 600-700 steps,

whereas without the bridging classifier it takes 1600-1700

steps. There are many other ways to implement "bridging"

classifiers. For example, the following classifiers also serve

as bridges for the sequences 1–9 and 11–19: 10,000 8000 6000 4000 2000, 800 1600 2400 CYCLE STEP 3200 4000 Figure 8: Marginal performance and the strength of classi-fier 1 observed when the system is run in the world shown in Figure 5, with and without "bridging" classifiers. the first one in the sequence. Thus the brigding classi-fier passes strength to all the classifiers in the chain rather than to just the first one. (Classifier 22 acts similary for the bottom sequence.) To test the effectiveness of the second kind of bridging classifier, the system was run in the world described in Figure 5 with the 18 classifiers described earlier and the bridging classifiers 21 and 22. The results were similar to the results obtained using the bridging classifiers 10 and 20: the strength of classifier 1 began to rise immediately, in the first 200 steps, as did the marginal performance. The system again reached 90% of its fixed-point performance rate in about 700 cycle steps. The main difference between using the second type of bridging classifier and the first is the fixed-point strengths of the classifiers in the top sequence. With the first type of bridging classifier, the fixed-point strength of classifier 1 was changed from 2000 (for μ(W 9) = 100) to 8000 (for μ(W 9) = 400); the fixed-point strengths of the other clas-sifiers in the top sequence changed from 1000 to 4000. With the second type of bridging classifier, the fixed-point strength of classifier 1 was changed from 2000 (for μ(W 9) = 100) to 8000 (for μ(W 9) = 400); the fixed-point strengths of the other classifiers in the to sequence changed from 1000 to values ranging from 7406 (for classifier 2) to 4000 (for classifier 9). The reason classifiers 2 through 9 have different fixed-point strengths is that the second type of 191 STRENGTH OF CLASSIFIER*!, WITH BRIDGE STRENGTH OF CLASSIFIER *| , NO BRIDGE MARGINAL PAYOFF TO SYSTEM, WITH BRIDGE MARGINAL PAYOFF TO SYSTEM, NO BRIDGE

Classifier 21 says "If the message list contains a message

posted by any of the classifiers in the top sequence, then

post a message". This classifier will be activated by classi-

fier 1 and remain active until the system receives a payoff

in state W 9 . The difference between the this type of epoch

marker and the one described earlier is that this one is ac-

tivated by every classifier in a sequence, rather than just e i , e i /m 10 i = l . . . 9 (21) e i , e i /m 20 i = 11...19 (22) 0 0

bridging classifier pays some of its strength to each classi-

fier in the sequence, but it pays more to the classifiers at

the beginning of the sequence, since that is when the bridg-

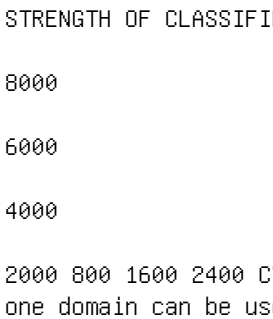
ing classifier's strength is the highest (just after receiving

a payoff). 5 SEQUENCES WITH MULTIPLE MESSAGE SOURCES In the experiments described in the previous section, each

classifier (except the first one) in a sequence was activated

solely by the message produced by the classifier that pre-ceded it in the sequence. Those sequences implemented something akin to a reflex: once the first classifier in the sequence is activiated by some signal from the environment, the rest of the classifiers in the chain are activated, one af-ter another, until the last one fires, no matter what effect their actions are having. Since each classifier in the se-quence used only messages from its predecessor, each paid its full bid to that predecessor, so that all classifiers in a sequence had the same fixed-point strength (ignoring the effects of bridging classifiers). Another type of classifier sequence is one in which one or more classifiers after the first one in a sequence have conditions that match messages produced by sources other than their predecessors in the sequence. For example, some of the classifiers could have one condition that matches a message produced by its predecessor and a second condition that matches a detector message produced by the system's input interface. Sequences of this type are non-reflex se-quences: the system can monitor the effects of executing each step, so that if the sequence isn't producing the ex-pected results (as indicated by messages on the message list), the sequence can be stopped or alternatives steps can be executed. To test the effectiveness of the bucket brigade algorithm in allocating strength down non-reflex chains, the CFSC/FSW1 system again was run in the finite state world shown in Figure 5. In these experiments the following nine classifiers were used for the top path: d 0 ,d 0 /e 1 ,r =

l (1) e s ,d s /e 4 , r= 1 (4) e 6 ,d 6 /e r ,r=l (7) e i _1 ,e i _1 /e i ,r = 1 i = 2,3,5,6,8,9 Classifier 1 is matched by state W 0 . Once it is activated, classifier 2 is activated by the message produced by clas-sifier 1, and then classifier 3 is activated by classifier 2's message. Classifier 4 is activated only if classifier 3 posted a message on the previous step and if the system is in state W s . Classifiers 5 and 6 then fire reflexively, and classifier 7 fires only if the system is in state W 6 . Classifiers 8 and 9 then fire reflexively. A similar set of classifiers (11 to 19) was included for traversing the bottom path. Classifier 1 and 11 compete when the Markov process is in state W 0 to guide the system down the top or bottom path, respectively. 192 Copyrighted Material Figure 9: Marginal performance and the strength of classi-fiers 1,4, and 7 when the system is run with two competing non-reflex sequences of classifiers, in which classifiers 4 and 7 each use one detector message and one message from their predecessors in the sequence. Figure 9 shows the results of running this set of clas-sifiers in the world shown in Figure 5, with the payoff for state W 19 = 100 and the payoff for state W 9 changed from 100 to 400 at step 0. The marginal performance (per 200 steps) is plotted against the major cycle step executed. The strength of classifier's 7, 4, and 1 are also plotted. As ex-pected, the strength of 7 begins to rise first, since it is closest in the sequence to the reward state, followed by the strength of classifier 4 and then 1. When the strength of classifier 1 begins to rise the marignal performance begins to rise, since the classifier 1 begins to win the competition with classifier 11 more often, thus leading the system down the top path to the higher payoff. Note, however, the fixed-point strength of classifier 4 is 1/2 that of 7, and strength of classifier 1 is in turn 1/2 that of 4. The reason for this drop is that classifiers 4 and 7 pay only 1/2 of their bid to their predecessor classifiers; the other half of their bids is paid to the system for the detector messages that match those classifiers. In general, then, any sequence that has n classifiers that use messages not from their predecessors in the chain will have an exponential (in n) fall-off in strength down the sequence. In this experiment this exponential fall-off didn't hinder the ability of the system to respond to the change in payoffs at the end of the sequences, since both competing sequences contain the same number of classifiers using messages from multiple sources. In other classifier structures, where the competing sequences have different number of classifiers 8000 6000 4000 2000 0 0 800 1600 2400 3200 4000 CYCLE STEP STRENGTH OF CLASSIFIER STRENGTH OF CLASSIFIER



for use in one domain can be used in other domains that are similar. For example, a group of classifiers (e.g., to control a robot's "hand") that were discovered while the system was learning to do one task also could be used to solve some other task, greatly reducing the time required to learn the second task. This "knowledge sharing" not only makes it possible to learn faster, it also leads to a more economical use of classifiers, since each situation will not require its own unique classifiers.

Figure 10: Marginal performance and the strength of classifier 1 when the system is run with two competing non-reflex sequences of classifiers, with and without "bridging" classifiers.

using messages from multiple sources, the results would be different.

Figure 9 also shows that the response time of the non reflex sequence was about the same as was observed with the reflex-like sequence (described in the previous section).

To see if bridging classifiers could speed up the learning rate in non-reflex sequences, the classifiers described above were run with the two bridging classifiers, 10 and 20, described in the previous section.

Figure 10 shows the results obtained for the non-reflex

sequence when run with and without bridging classifiers.

As with the reflex-like sequences, the rate of learning with

the non-reflex sequence is greatly increased by the use of

a bridging classifier like classifier 10. With the bridging

classifier, the strength of classifier 1 and the system's per-

formance began to increase immediately, in the first 200

cycles, and it reached 90% of its fixed-point performance

rate in about 700 cycles. Also note that the fixed-point

strength of classifier 1 is now 5000: the strength passed

through the bridging classifier fa able to overcome the ex-

ponential fall-off of strength observed in non-reflex chains

without bridging classifiers.

6 SEQUENCES WITH SHARED PARTS

Because classifier systems exchange information through a

fully open "blackboard", the message list, any classifier can

be used in any context, as long as the message list contains

messages that satisfy the classifier's conditions. One advan-

tage of this architecture is that classifiers that are created Figure 11: Paths leading to rewards at states W 4 and W 8 . In order to explore the effect of shared classifiers on the allocation of strength by the bucket brigade algorithm, the CFS-C/FSW1 system was run in the simple finite state world shown in Figure 11. There are 9 states, in this world. The start state is W 0 . When the system fa in W 0 , if the classifiers set the effector value r to 1, then the system goes to state W 1 with probability 1; if the classifiers set r to 2, then the system goes to W 8 . Once the top path fa chosen, the Markov process can be moved through the intervening states to W 4 by continuing to set r to 1. If the bottom path fa

chosen, in order to move to W 8 the system must first set r to 1 to get to W 6 and then set r to 2 for the rest of the bottom path. When the process reaches to state W 4 or W 8 , it always returns to state W 0 . State W 4 has a payoff of 100; all other states have 0 payoff. The CFS-C/FSWl system was run in the world shown in Figure 11 with the following 8 classifiers: d 0 ,d 0 /e 1 ,r = 1 (1) e i _ 1 ,e i _ 1 ,/e i ,r = 1 (i = 2, 3, 4) d 0 ,d 0 /e 5 ,r = 2 (5) e 5 ,e 5 /e 6, r = 1 (6) e i _ 1 ,e i _ 1 ,/e i ,r = 2 (i = 7, 8) Classifiers 1 and 5 compete to select the path to be followed, i.e., 1 selects the top path and 5 selects the bottom. Once a path is chosen the rest of the classifiers in each sequence (2 -4 or 6-8) are executed in order. Note that there are no classifiers shared between the two sequences in this case. The system was also run with classifiers similar to those shown above, but with classifiers 2 and 6 replaced by the following single classifier: 193 CLASSIFIER CLASSIFIER '1 , BRIDGE '1,N0 BRIDGE PAYOFF, BRIDGE PAYOFF, NO BRIDGE e 2 , 5 ,e 1 , 5 /e 9 ,r=l (9) r=2 μ = 100 r=1 r=2 r=2 r=1 r=l r=1 r=1 W 0 w 1 w 2 w 3 W 4 W 5 W 6 w 7 w 8 0 0 When no classifiers are shared, the system basically achieves the optimal fixed-point performance (4000 per 200 steps). However, when a classifier is shared the system's perfor-mance is the about what it could achieve by choosing the path at random (2000). The performance with shared classifiers is consistent with the fixed-point strengths of classifiers 1 and 5, which are about the same. When those classifiers compete to se-lect a path, they each win 50% of the time. As a little thought will show, the reason classifiers 1 and 5 have the same strength when classifier 9 is shared between the chains is obvious: both 1 and 5 are paid only from classifier 9, and the fixed-point strength of classifier 9 is just the average of the payments made to it whenever it used. Adding a bridging classifier rectifies this situation: clas-sifier 1 now gets income from both classifier 9 and its bridge classifier, 10, and classifier 5 gets income from classifier 9 and its bridge, classifier 11. Since the bridging classi-fier 10 gets the 100 payoff, while classifier 11 gets 0, the strength of classifier 10 is greater than that of 11, and in turn the strength of classifier 1 is greater than that of 5. Thus adding a bridging classifier restores the fixed-point performance rate to near the maximum. 7 CONCLUSIONS The apportionment of credit problem is the problem of deciding, when many rules are active at every time step, which of those rules active at step t are necessary and suf-ficient for achieving some desired outcome at step t + n. In classifier systems using the bucket brigade

algorithm, credit is allocated in the form of a value, strength, associated with each classifier. Because the bucket brigade algorithm uses only local information to incrementally change the strength of classifiers, one key problem for systems using the bucket brigade algorithm is how rapidly strength can be passed down long sequences of classifiers. If a whole sequence of classifiers must be activated many times in order to adjust the strength of a classifier at the beginning of the chain in response to a change in payoff associated with the last step in the sequence, the system's response to simple changes in its environment may be too slow to be useful. This paper has presented results that show the bucket brigade basically works as designed—strength is passed down a chain of coupled classifiers from those that receive the reward directly from the environment to those that are "stage setters". The system can thereby learn to respond to a change in the environment by choosing to activate one classifier rather than another, even when a reward is re-ceived only after a long sequence of classifiers is activated. However, it does seem to take a large number of trials before the classifiers at the beginning of a chain reach their fixed-point strengths, and so alter the choice of paths to take. While increasing the bid constant k can speed the flow of strength, a large k means classifiers are risking a large proportion of their strength on each bid they make, so that there is not much room for classifiers to make mistakes. Increasing the BidPow parameter was shown to increase the fixed-point payoff rate the system will achieve, and to slightly decrease the systems response time to a change in payoff at the end of the sequence. The bucket-brigade algorithm was shown to allocate strength down sequences that implement both "reflex" like subroutines and non-reflex like sequences. The system can respond to changes in payoff at the end of two competing non-reflex sequences just as fast as it can to changes that occur at the end of two competing reflex sequences. How-ever, the fixed-point strengths of classifiers in non-reflex se-quences fall off exponentially with each classifier that uses a message not from its predecessor in the sequence. This drop in strength could present problems for non-reflex se-quences that try to compete with reflex-like sequences, and it could have effects on the creation and deletion of rules if strength is used to bias the rule discovery algorithms used by the system. One way to ameliorate the fall of strength in non-reflex chains that use only detector messages is to change the bucket brigade algorithm so that classifiers pay less than the full BidShare for detector messages. This would mean that classifiers that match detector

messages will in gen-eral have higher fixed-point strengths than other classifiers. The higher fixed-point strengths may bias the system's rule discovery algorithms to create more detector messages—a bias which makes some sense, since the system should put a high priority on using messages from its environment. On the other hand, such a bias in favor of classifiers that use detector messages won't solve the exponential strength 194 No Sharing Shared Classifier Shared, with Bridge 3880 1990 3913 s fp ,1 2048 1075 6035 S f p ,5 651 1065 1677 Classifier 10 serves to bridge the top sequence, and classifier 11 serves to bridge the bottom sequence. All classifiers were started with a strength of 1000. The following table shows the results obtained for the three sets of classifiers: (e 1 | m 10), ~ d 4 /m l0 (10) (e 5 | m 11), ~ d 8 /m 11 (11) Classifier 9 is shared by the two sequences: it matches mes-

sages produced by either classifier 1 or classifier 5, and sets the effector value r to 1. (Classifiers 3 and 7 were modi-fied so they both respond to to the message produced by classifier 9; "pass-through" symbols were used to ensure that classifiers 3 and 7 fire only when classifiers 1 and 5, respectively, were fired two steps before.) The set of classifiers with the shared classifier, 9, was

also run with the following bridging classifiers:

fall-off problem for non-reflex sequences that use messages

from other classifiers.

"Bridging" classifiers were shown to have many effects

on the allocation of strength and the performance result-

ing from competing sequences of classifiers. Bridging clas-

sifiers lead to a dramatic decrease in the number of passes

the system must make down a sequence before it can re-

spond to a change in the environment, for both reflex and

non-reflex sequences. Bridging classifiers also allocate ad-

ditional strength to the earlier classifiers in non-reflex se-

quences, overcoming the exponential fall-off of strength

seen without such bridging classifiers.

Note that classifier sequences with bridging classifiers require the same number of passes down the chain to respond to a change in payoffs at the end of the sequence, no matter how long the sequence is. For example, when the experiments described in Section 4 were repeated with sequences of 19 classifiers, the system's performance with bridging classifiers again began to increase almost immediately, within the first 15 -20 trials, just as it did with sequences of 9 classifiers. Without bridging classifiers, the length 9 sequence required about 120 trials to respond, whereas the length 19 sequence required 250 trials.

One way to decrease further the response time might be to use multiple bridging classifiers for each sequence. Each bridge would pass additional strength to the first classifier in the sequence, enabling it to dominate the competition sooner.

There are many ways to implement bridging classifiers. Two ways were tried in experiments described in this paper. While each type of bridging classifier acted to decrease the system's response time, each also resulted in somewhat different distributions of strength over the classifiers in a sequence. These differences in the fixed-point strengths

may have important effects on long term learning in the system if strengths are used to guide the creation and deletion of classifiers. Other types of bridging classifiers should be tried to see what effects they have.

Finally, sharing clasifiers between sequences could be a very good way to promote transfer of knowledge from one domain to another, and to economically use the same rules in more than one context. However, experiments described in this paper show that sharing classifiers can lead to problems for the allocation of strength down sequences of classifiers by the bucket brigade algorithm. Basically, sharing classifiers means the information being passed from classifier to classifier (in the form of strength) is lost, or at least greatly attenuated, when shared classifiers are involved. Simple bridging classifiers were shown to be one

Learning systems that operate in environments with huge numbers of states must be able to categorize the states into equivalence classes that can be treated alike. Holland-type classifier systems can learn to categorize states by building default hierarchies of classifiers (rules). However, for default hierarchies to work properly classifiers that implement exception rules must be able to control the system when they are applicable, thus proventing the default rules from making mistakes. This paper presents results that

show the standard bucket brigade algorithm does not lead to correct exception rules always winning the competition with the default rules they protect. A simple modification to the bucket brigade algorithm is suggested, and results are presented that show this modification works as desired: default hierarchies can be made to achieve payoff rates as near to optimal as desired. 1 INTRODUCTION

Any learning system that is to operate in environments with huge numbers of states must be able to categorize the states into equivalence classes that can be treated alike. For rule-based systems like classifier systems ([Holland, 1986a], [Burks, 1986], [Holland and Burks, 1987]) the problem involves finding a set of classifiers (condition/action rules) that induce the appropriate equivalence classes. One approach to this problem is to try to find a set of rules that never make mistakes and that partition the whole environment. Such a set of rules in effect establishes a homomorphic model of the world. The problem with this approach is that realistic environments typically involve millions of possible states, with very complicated underlying equivalence classes, of which the system may have sampled only a small fraction. In such situations it would take a vast number of rules to establish a homomorphic model of the world. Another approach is to implement a default hierarchy ([Holland, 1985], [Holland, 1986b]) of

rules. A default hier-

archy is a multi-level structure in which classifiers (rules) at

the top levels are very general. Each general rule responds to broad set of states, so that just a few rules can cover all possible states of the world. Of course since a general rule responds in the same way to many states that don't really belong in the same category, it will often make mistakes. To correct the mistakes made by the general classifiers, lower level, exception rules are added to the default hierarchy. The lower level classifiers are more specific than the higher level rules: each exception rule responds to a subset of the situations covered by some more general rule. Default hierarchies have several features that make them well suited for learning systems that must build models of very complex domains: • Default hierarchies can be made as error-free as nec-essary, by adding classifiers to cover exceptions to the top level rules, to cover exceptions to the exceptions, and so on, until the required degree of accuracy is achieved. • Default hierarchies are the basis for building quasihomomorphic models of the world, which generally require far fewer rules to implement a given degree of accuracy than do equivalent homomorphic models [Holland, 1986b]. • Default hierarchies make it possible for the system to learn gracefully, since adding rules to cover excep-tions won't cause the system's performance to change drastically, even when the new rules are incorrect. This paper describes some simple experiments with de-fault hierarchies implemented in the CFS-C/FSW1 classi-fier system [Riolo, 1987b]. These experiments show that when default hierarchies are built in a top down manner, by adding rules to cover exceptions, overall performance does improve as predicted. However, using the standard bucket brigade algorithm, the system does not achieve the performance expected. The reasons for this lower than ex-pected performance are explained, and a modification to the bidding mechanism used in the standard bucket brigade algorithm is proposed. This modification is shown to lead to performance as close to the expected performance as desired. *This work was supported by National Science Foundation Grant DCR 83-05830

2 THE CFS-C/FSW1 SYSTEM

All experiments described in this paper were done using

the CFS-C classifier system [Riolo, 1986], set in the FSW1

("Finite State World 1") task environment [Riolo, 1987a].

This section briefly describes the parts of the CFS-C/FSW1

system that are relevant to the experiments described in

this paper. For more details, see [Riolo, 1987b] or the cited

documentation.

Basically, the FSW1 domain is a world that is mod

eled as a finite Markov process, in which a non-zero payoff

is associated with some states. The classifier system's in

put interface provides a message that indicates the current

state of the Markov process. The classifier system's effec

tor interface provides the system with a way to alter the

transition probabilities of the process, so that the system

can control (in part) the path taken through the finite state

world. Basically, a message that begins with "10" is inter

preted by the effector interface as a command to set the

effector to some value r, r = 0 ,1 , . . . 15, depending'on the

right most 4 bits of the message. The effector setting r

is used to select the transition matrix P(r) which specifies

the probability the system will go from the current state W i

to some state W j . When the system moves to a new state

with a non-zero payoff, that payoff is given to the active

classifiers as a "reward".

The CFS-C classifier system is a standard, "Holland"

type learning classifier system. While the CFS-C classifiers

all have two conditions, in this paper they will be treated

as if they have only one condition. (This is done by making

both conditions identical.) For the experiments described in this paper, no classi

fiers are coupled, i.e., no classifier is satisfied by a message

produced by another classifier. All classifiers match only

detector messages, i.e., the classifiers are matched when the

current state of the finite state process is in the set of states

matched by the classifier's condition. The most important

parts of the classifier system are (a) the mechanism imple

menting bidding and competition to post messages and to

set the effectors, and (b) the allocation of payoff to classi

fiers from the environment. In the CFS-C/FSWl system the bid of classifier i at

step t, Bi(t), is calculated as follows: B i (t) = k∗ S i (t) ∗ BidRatto i

k is a small constant (set to 0.1), which acts as a "risk

factor", i.e., it determines what proportion of a classifier's

strength it will bid and so perhaps lose on a single step.

Si(t) is the strength of classifier i at step t. BidRatioi is a

number between 0 and 1 that is a measure of the classifier's

specificity, i.e., how many different messages it can match.

A BidRatio of 1 means the classifier matches exactly one

message, while a BidRatio of 0 means the classifier matches

any message. Thus classifiers that implement high level,

general rules in a default hierarchy will have low BidRatios, 197 while classifiers that implement lower level, exception rules will have higher BidRatio values. When a competition is run to determine which classi-fiers are to be activated and post messages, the probability that a given (satisfied) classsifier i will win is: Prob(i wins)=β i (t)/Σβ i (t) i where j ranges over all bidding (satisfied) classifiers at t, and β i (t), the effective bid of classifier i at t, is: β i (t) = B i (t) BidP ° w BidPow is a parameter that can be set to alter the shape of probability distribution used to choose classifiers to pro-duce messages. In all experiments described in this paper, BidPow = 3. Note that the output interface of the CFS-C/FSW1 sys-tem may have to resolve conflicts, e.g., when one classifier produces a message that says "set the effector value r to 1" and another produces a message that says "set the effector value r to 2". Since the effector can only be set to one value at a time (just as we can either lift our arm or lower it, and not both), an effector conflict resolution mechanism must be used. Basically, the value of r is chosen probabilistically, with the probability that r = r' equal to: Σ m' β m' (t)/Σ m β m (t) where m' ranges over the effector messages that say "set r to r'", β m' (t) is the effective bid of the classifier that posted message m', and m ranges over all effector messages. Once an effector value is chosen, all messages that are inconsis-tent with that setting are deleted from the new message list. When a classifier wins a competition and posts a mes-sage, its strength is decremented by the amount it bid to become active. Since there are no coupled classifiers, classi-fiers only receive payments from the environment when the system moves to a state with an associated non-zero pay-off. In particular, the full payoff is added to the strength of every classifier that has posted a message during that time step. Because the bids made by classifiers are not paid to other classifiers in the experiments described in this paper, the strength at step t + 1 of a classifier i that produced one or more messages at step t is: S i (t + l) = S i (t) - B i (t) + R (t) where B i (t) is the classifiers bid at step t and R(t) is the reward from the enviroment at step t. Note that the fixedpoint strength of classifier i, S i,jp , is inversely

proportinal to its bidratio, BidRatio i . In particular, S i,jp = I i /(k * BidRatio i) where I i is the average amount paid to classifier i whenever

it posts messages. Other things being equal, a general clas-

sifier (with a low BidRatio) will have a higher fixed-point

strength then a more specialized classifier (with a higher

BidRatio). d 0/1 /r = 1 (1) Figure 1: A simple FSW1 finite state world. 3 A SIMPLE T E S T WORLD In order to examine default hierarchies in the CFS-C/FSWl system, consider the simple FSWl world shown in Figure 1. There are seven states in this world, W i , i = 0 . . .6 . State W 5 is the start state. States W 4 and W 6 have pay-offs of 200 and 400, respectively; all other states have zero payoff. For i = 4, S, or 6, and any effector value r, Pij{r) = 0.25, j = 0 . . . 3 . That is, when in state W 4 , W 5 , or W 6 , the system has an equal chance of going to any one of the four states on the left in Figure 1, and no chance of going to a state on the right, no matter what the value of r. When in state W 0 or W 1 , if r = 1 the system goes to W 4 ; if r = 2, the system goes to W 5 . When in state W 2 or W 3 , if r = 2 the system goes to W 4 ; if r = 1, the system goes to W 5 . And when in state W 3 , the system goes to W 6 only if r = 3. (Other values of r are not allowed in the experiments described below.) The system can do best if it sets r to 1 when in states W 0 or W 1 , sets r to 2 when in state W 2 , and sets r to 3 when in state W 3 . A good set of classifiers for this world would classify the states on the left into three categories and respond accordingly. (Since the system can't alter the transition probabilities when in any of the states on the right, no rules categorizing those states can be more use-ful than any others.) While this world is too small to show how a default hierarchy (i.e., a quasi-morphism) can require fewer classifiers than a homomorphic model, it is complex enough to show the relationship between default and ex-ception rules. As. a simple test of the CFS-C/FSWl system, it was run in the world shown in Figure 1 using the following three classifiers: (The numbers in parentheses on the right serve to iden-tify the classifiers.) The condition "d 0/1 " matches two de-tector messages, for states W 0 or W 1 The conditions d 2 and d 3 each match one detector message, i.e., the detector messages for states W 2 and W 3 , respectively. The action "r = x" posts an effector message that sets the effector value r to x, x = 1, 2, or 3. In this and other experiments described in this paper, the

system was run for 2000 major-cycle steps. In all runs classifiers reached their fixed-point strengths within the first 1000 steps. The average performance of the system over steps 1500 to 2000 is used to establish the "fixedpoint" performance, P fp , for one run. All results presented are the average of 10 runs (each started with a different pseudo-random number generator seed). For the three classifiers shown above, P fp was 124.4 per step (StdDev = 1.3). As can be easily calculated, the expected value is 125 (the system gets a non-zero payoff at most once every two steps). Thus these three rules perform just as expected. Of course these rules do not implement a default hierarchy: there is a general rule, 1, but it never makes mistakes, and the other rules do not cover subsets of the cases covered by the general rule. Instead, these rules implement a homomorphic model, since the rules partition the space of possible states (with respect to the 4 states on the left in Figure 1, the states the system is modeling). The next section shows how a default hierarchy can be built to cover the same states. This general classifier clearly implements a high level de-fault rule for this world, namely "When in any state W i, i = 0 . . . 3 , set r to 1". The expected fixed-point performance for a system using just this rule is 50, since the system will get a reward only when it moves from state W 0 or W 1 to W 4 , which it will do 50% of the time. The observed fixed-point performance was 49.9, or 99.8% of the expected value (StdDev = 1.9%). The classifier's fixed-point strength is 1765, which is also exactly the expected strength. To improve the system's performance, it was run again with two rules, the above default rule, 4, and the following "exception" rule: 198 w 3 w 2 W 0 W 1 r=1 r=2 w 4 r=2 w 5 r=1 r=3 W 6 d 2/3 /r = 2 (BidRatio = 0.75) (5) 4 RESULTS To test the performance of a default hierarchy in the world shown in Figure 1, the CFS-C/FSW1 system first was run with just one classifier: do|1|2|3/r = 1 (BidRatio = 0.56) (4) d 3 /r = 3 (3) d 2 /r = 2 (2)

This classifier covers a subset of the states covered by clas-

sifier 4. For that subset, it corrects a mistake made by

classifier 4, since it sets r to 2, and so causes the system

to go from state W 2 or W 3 to W 4 (instead of to W 5). The

expected payoff for a system using the default hierarchy

implemented by these two rules is 100 per step, since every other step the system should receive the 200 payoff from state W 4 . The observed fixed-point payoff was 83.9, or just 83.9% of the expected rate (StdDev = 1.8%). The fixed point strengths of classifiers 4 and 5 were 2661 and 2667, respectively. Note that the strength of the default rule 4 went up as predicted [Holland, 1985] when the exception rule 5 is added to the system, but it did not go up as high as expected (to 3571).

The system also was run with an additional classifier: d 3 /r = 3 (BidRatio = 1) (6)

This rule covers another exception, so that the system can get to the high-payoff state W 6 whenever it is in state W 3 .

Together rules 4, 5, and 6 implement a complete default hierarchy for this simple world. The expected payoff for these classifiers is the same as for the original 3 perfect rules (1, 2, and 3), i.e., 125 per step. However, the observed performance was just 108.7, or 86.9% of the expected value (StdDev = 1.3%). The fixed-point strengths of classifiers 4, 5, and 6 were 2860, 2667, and 4000, respectively.

Why is the performance lower than expected? First, note that the fixed-point strengths of classifiers 5 and 6 are just as expected, given the amount of payoff they receive. On the other hand, classifier 4, the top level

default

rule, has a fixed-point strength that is much lower than

its expected value, 3571. Thus classifier 4 must be making

mistakes. To get a better idea of what is happening consider Fig-

ure 2, which shows the results of a run using just two clas-

sifiers, 4 and 5. Figure 2 shows the marginal performance

of the system plotted against major-cycle steps executed.

It also shows the strengths for classifiers 4 and 5. First, note that the strength of classifier 5 stabilizes at

2667, which is the maximum strength it can reach given its

maximum average income (200 per bid) and its BidRatio,

0.75. On the other hand, the strength of classifier 4 oscil-

lates, and it never reaches its expected value. Instead, as

soon as the strength of 4 gets much above the strength

of classifier 5, performance begins to drop (as does the

strength of classifier 4). This co-oscillation of the strength

of classifier 4, the general default rule, and marginal per-

formance is the key to why the performance of this default

hierarchy is lower than expected. Recall that in classifier systems there is a competition to

post messages and control effectors: the higher a classifier's

bid, the more likely it is to win the competition and control

the system's behavior. Also recall that when a classifier

loses the competition to set an effector, its messages are

deleted from the message list and it does not pay its bid to CYCLE STEP Figure 2: Performance of a default hierarchy using the stan-dard bucket brigade algorithm. its suppliers. Bearing these facts in mind, the reason for the oscillations and sub-expected performance are clear: 1. When the strength of an exception rule (rule 5) is greater than the strength of the default rule for which it is exception (rule 4), the exception rule tends to control behavior in the states it covers. The exception tends to protect the default from making mistakes, which allows the average income for the default rule to reach its maximum. 2. Since the default rule has a smaller BidRatio than the exception rule that protects it, if the average in-comes of the rules are about the same, the maximum fixed-point strength of the default rule will be greater than the maximum for the exception rule. In this ex-ample, as rule 5 protects 4, the strength of rule 4 eventually exceeds that of rule 5 (since both have a maximum income of 200 per bid). 3. As the strength of the default classifier rises, its bid and effective bid will rise. Once the effective bid of the default rule is near to or greater than the effective bid of the exception rule, the default rule will begin to win the competition. Tha t is, the exception rule will not always win the competition in the situations it covers, which means it will not be able to stop the default rule from making mistakes. 4. Once the default rule begins making mistakes, both the system performance and the strength of the de-fault rule will begin to fall. Eventually the system re-turns to the beginning of the performance oscillation, when the strength of the default rule is enough lower than tha t of the exception rule so that the exception 199 MARGINAL PAYOFF TO SYSTEM STRENGTH OF DEFAULT CLASSIFIER (# 1) STRENGTH OF EXCEPTION CLASSIFIER (# 2) 6000 5000 4000 3000 2000 1000 0 0 500 1000 1500 2000 rule can again protect the default from making mis takes.

In short, the protection an exception-covering rule provides

for a default rule allows the strength of the default rule to rise until the exception rule can no longer protect it. This will almost always happen in default hierarchies, since the maximal fixed-point strength of a general default classifier is in general higher than that of of more specific, exception classifier. One way to correct this problem would be to alter the bucket brigade so that the maximum fixed-point strength of general classifiers is not higher than that for more special ized classifiers. For example, payments to classifiers could be biased so that a lower BidRatio leads to a smaller share of the

payment from other classifiers or from environmen tal rewards. Lowering the fixed-point strength of general classifiers may create some other problems, however. For instance, since default rules tend to make mistakes more often than more specialized rules, the fixed-point strength of a default rule should be relatively high, so that it can afford to make mistakes (and lose strength) without hav ing its strength go so low that it its eliminated from the system. Another approach is to leave the relationship between the fixed-point strengths of general versus specific classi fiers the same, but to bias the effective bid, 0, against the general classifiers. Since the effective bid only changes the probability distribution used to determine which classifiers post messages and control the systems effectors, it does not alter a classifier's per bid income or payment. Thus general classifiers will still have relatively high fixed-point strengths, even though specialized classifiers will tend to win the competition more often. To test this idea, the way effective bids are calculated in the CFS-C/FS Wl system was changed by adding a factor involving a classifier's BidRatio. In particular, the effective bid, β i (t), of a classifier i at step t is now calculated as follows: β i (t) = B i (t) BidPow ∗ BidRatio i EBRPow The value of EBRPow can be changed to alter the shape of probability distribution used to choose the classifiers that are to produce messages. For example, if BidPow = 1 and EBRPow = 0 then β i (t) = B i (t), i.e., a classifier's probability of producing messages is just its bid divided by the sum of bids made by all satisfied classifiers. Setting EBRPow equal to 1, 2, and so on, makes it less likely that general classifiers (those with lower BidRatio's) will win the competition with more specific, co-active classifiers. To test the effects of this modification, the system was run in the same finite state world described earlier, using classifiers 4, 5, and 6, which implement a simple 3 level default hierarchy. The following table shows the results obtained using various values for EBRPow: % Expected EBRPow P fp (1 StdDev) S 4,fp 0 86.9 (1.3) 2860 1 92.7 (1.3) 3104 2 95.8 (0.9) 3280 4 98.6 (1.1) 3479 6 99.5 (1.8) 3553 As can been seen, raising the EBRPow parameter increases the average fixed-point performance of the system to vir tually a mistake-free level. Also note that the fixed-point strength of classifier 4, the default rule, increases to almost its maximum value, 3571. CYCLE STEP Figure 3: Performance of a default hierarchy using the modified bucket brigade algorithm with EBRPow = 3. Figure 3 shows the results of repeating the experiment shown in Figure 2 with EBRPow = 3. The oscillations of both the marginal

performance and the strength of classi fier 4 are all but eliminated. Thus with a high EBRPow, the exception classifier 5 is able to almost always win the competition with the default rule it protects, in which case the system never makes mistakes. The system was also run using the same three classi fiers in two slightly different versions of the world shown in Figure 1. First, the system was run in a world in which setting the effector value r to 2 in states W 2 and W 3 causes the system to go to another state, W 7 which has a payoff of 100. That is, an exception rule like classifier 5 leads to a lower payoff than the default rule does when it doesn't make a mistake. In this world the expected performance (per step) is 112.5, while the observed performance with EBRPow = 0 was 97.7 (86.9% expected performance, StdDev = 1.8). However, with EBRPow = 3 the ob served performance was 109.0 (97.5% of the expected value, StdDev = 1.9). Thus the default hierarchy performs just 200 0 500 1000 1500 2000 6000 5000 4000 3000 2000 1000 MARGINAL PAYOFF TO SYSTEM STRENGTH OF DEFAULT CLASSIFIER (# 1) STRENGTH OF EXCEPTION CLASSIFIER (# 2)

as well when the exception rule leads to a lower payoff than

the default rule does when it is correct. Second, the system was run in a finite state world like

that shown in Figure 1, except that various amounts of un-

certainty are introduced into the transitions. For example,

instead of going from state W 0 to state W 4 with probability

1.0 when r is set to 1, the system will go to that state with

probability 0.92 and go to one of the other states on the

right with probability 0.04 each; i.e., 8% of the time the

system will go somewhere unexpected. The following table

shows the results of running the three classifiers 4, 5, and

6 that implement the simple default hierarchy described

earlier, in worlds with increasing amounts of uncertainty

(using EBRPow = 3): % Un- Expected Observed % Optimal

certainty Payoff Payoff (1 StdDev) Payoff 0 125 122 (3.44) 97.6 8 116 115 (3.64) 99.1 16 107 105 (3.64) 98.1 32 89.0 87.5 (5.52) 98.3

As can be seen, uncertainty in the environment has little effect on the systems ability to obtain the best possible payoff rate. This is an important property for any learning system that must contend with very complex environments in which it can never completely reduce the uncertainty.

5 CONCLUSIONS

Default hierarchies are an excellent way for classifier systems to cope with very complex enviroments. However, for default hierarchies to work properly, classifiers that imple-ment "exception" rules must be able to control the system when they are applicable, thus protecting the default rules from making mistakes. This paper has presented results that show the bucket brigade algorithm as described in [Holland, 1985] does not lead to correct exception rules always winning the competition with the default rules they protect. A simple modification to the bucket brigade algorithm is suggested, which involves biasing the calculation of the effective bid so that general classifiers (those with low BidRatios) have much lower effective bids than do more specific classifiers (those with high BidRatios). Results are presented that show this modification works as desired: de-

fault hierarchies can be made to achieve payoff rates as near

to optimal as desired.

Since the modification to the bucket brigade algorithm

only changes the effective bid made by classifiers, the allo-

cation of strength under the bucket brigade is not altered

(except insofar as the fixed-point strengths of default rules

are increased to their maximum expected levels). Also, if

the system does not have more specialized exception classi-

fiers to compete with a particular default rule, that default

rule will continue to control the behavior of the system.

adaptation to the task environment, Ph. D. Thesis, Analysis of the behavior of genetic adaptive systems, Ph. D. Thesis, machine learning, Ph. D. Thesis, Dept. Civil Eng., payoff, see fig. 2), this does no harm sinc

References

[1] Booker, L.B. 1985, "Improving the Performance of Genetic Algorithms in Classifier Systems". Proceedings of an International Conference on Genetic Algorithms and Their Applications, Pittsburgh, Pennsylvania : Carnegie-Mellon University.

[2] DeJong, K.A. 1975, "Analysis of the Behavior of a Class of Genetic Adaptive Systems". PhD Dissertation, Ann Arbor : The University of Michigan.

[3] Farmer, J.D., Kauffman, S.A., and Packard, N.H. 1987, "Autocatalytic Replication of Polymers". Physica 22D, pp. 50-67.

[4] Farmer, J.D., Kauffman, S.A., Packard, N.H., and Perelson, A.S. 1987, "Adaptive Dynamic Networks as Models for the Immune System and Autocatalytic Sets". Annals New York Academy of Sciences.

[5] Farmer, J.D., Packard, N.H., and Perelson, A.S. 1986, "The Immune System, Adaptation, and Machine Learning". Physica 22D, pp. 187-204.

[6] Feldman, J.A. 1986, "Neural Representation of Conceptual Knowledge". TR189, Dept. of Computer Science, Rochester, NY : The University of Rochester.

[7] Goldberg, D.E. 1983, "Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning". PhD Dissertation, Ann Arbor : The University of Michigan.

[8] Holland, J.H. 1975, Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan : The University of Michigan Press.

[9] Holland, J.H. 1985, "Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule Base Systems". Machine Learning II, Ch. 20, Los Altos, CA : Morgan Kauffman.

[10] Jerne, N.K. 1974, "Toward a Network Theory of the Immune System". Annals of Immunology (Inst. Pasteur) 125C : 373.

[11] Kaplan, S. 1986, Neural Models, Course Notes, Ann Arbor : The University of Michigan.

[12] Langton, C.G. 1984, "Self-Reproduction in Cellular Automata". Physica 10D, pp. 135-144.

[13] Rosenstein, R.W., Musson, R.A., Armstrong, M.Y.K., Konigsberg, W.H., and Richards, F.F. 1972, "Contact Regions for Dinitrophenyl and Menadione Haptens in an Immunoglobulin Binding More Than One Antigen". Proceedings of the National Academy of Science, USA, Vol. 69, No. 4, pp. 887-881.

[14] Wysocki, L.J., Manser, T., and Gefter, M.L. 1986, "Somatic Evolution of Variable Region Structures During an Immune Response". Proceedings of the National Academy of Science, USA, Vol. 83, pp. 1847-1851.

An Adaptive Crossover Distribution Mechanism for Genetic Algorithms

J. David Schaffer
Amy Morishima

Philips Laboratories
North American Philips Corporation
Briarcliff Manor, New York

ABSTRACT

This paper presents a new version of a class of search procedures usually called genetic algorithms. Our new version implements a modified string representation that includes special punctuation used by the crossover recombination operator. The idea behind this scheme was abstracted from the mechanics of natural genetics and seems to yield a search procedure wherein the action of the recombination operator can be made to adapt to the search space in parallel with the adaptation of the string contents. In addition, this adaptation happens "for free" in that no additional operations beyond those of the traditional genetic algorithm are employed.

We present some empirical evidence that suggests this procedure may be as good as or better than the traditional genetic algorithm across a range of search problems and that its action does successfully adapt the search mechanics to the problem space.

1. Background

A genetic algorithm is an exploratory procedure that is able to locate high performance structures in complex task domains. To do this, it maintains a set (called a population) of trial structures, represented as strings. New test structures are produced by repeating a two-step cycle (called a generation) which includes a survival-of-the-fittest selection step and a recombination step. Recombination involves producing new strings (the offspring) by operations upon one or more previous strings (the parents). The principle recombination operator was abstracted from knowledge of natural genetics and is called crossover. Holland has provided a theoretical explanation for the high performance of such algorithms [5] and this performance has been demonstrated on a number of complex problem domains such as function optimization [3, 4] and machine learning [6, 8, 9].

The action of the traditional crossover is illustrated in figure 1. Starting with two strings from the population†, a point is selected between 1 and L-1, where L is the string length. Both strings are severed at this point and the segments to the right of this point are switched. The two starting strings are usually called the parents and the two resulting strings , the offspring. Taking the metaphor one step further, we will call this operation a mating with a single crossover event. In all previous work with which we are familiar, the crossover point is chosen with a uniform probability distribution.

The motivation for our new crossover operator sprang from some properties of this traditional operator. Specifically, it has a known bias against properly sampling structures which contain coadaptive substrings that are far apart. In metaphorical terms, we might call them genes which are far apart on the chromosome. The reason is not difficult to grasp intuitively. The farther apart the genes are, the higher the probability that a uniformly selected random crossover point will fall between them causing them to be passed to different offspring. We observe that this crossover operator requires only knowledge of the string length; it pays no attention to its contents. Furthermore its action is nonadaptive. It performs the same way in every generation.

In contrast to this, what we know of Nature's genetic crossover activity suggests that the location of crossover events may be quite sensitive to the contents of the chromosome [1]. There are many activities in this microworld which involve the initiation of an action by the binding of an enzyme to a specific base-sequence. We were motivated to design a crossover mechanism which would adapt the distribution of its crossover points by the same survival-of-the-fittest and recombination processes already in place. We reasoned that it should do so by the use of special punctuation marks introduced into the string representation for this purpose. The operation envisioned for this new crossover was to proceed by "marking" the site of each crossover event in the string in which it occurred. Thus if the search space had the characteristic that crossovers at particular loci were consistently associated with inferior offspring, then they would die out, taking their markings with them. The converse was also hoped for. If no consistent relation existed to be exploited, then a

† We will show all strings as bit strings, but this is not a requirement imposed by the algorithm.

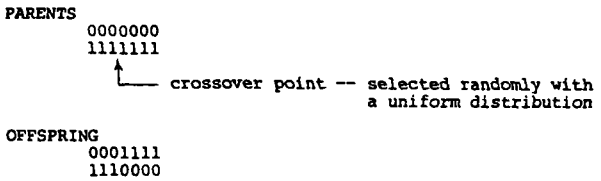


Figure 1. The action of the traditional crossover operator. The two parent strings are shown as all zeros and ones for clarity.

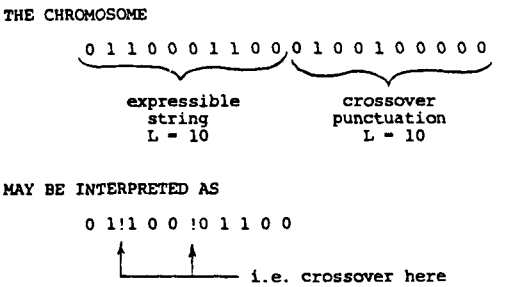


Figure 2. The string representation of chromosomes with crossover punctuation marks.

random allocation of markings was expected. In this, we were reminded of speculation by Lenat that modern gene pools may contain accumulated heuristics to guide genetic search as well as an accumulation of well-adapted expressible genes [7].

The idea of punctuation in the strings used by a genetic algorithm was first proposed by Holland [5], but this proposal was for a different purpose than that proposed here.

The rest of this paper will present, the mechanics of our new crossover operator, some empirical results indicating that superior performance can indeed be achieved with it, and some data exhibiting properties of its behavior.

2. Mechanics

In this section we will explain the mechanics of operation of our new crossover operator. We address how the crossover punctuation is coded, how an initial distribution is generated, how this distribution affects the crossing over of the functional parts of the strings, how the punctuation marks themselves are passed on to the offspring, and how a linkage is established between individual punctuation marks and the functional substrings with which they are associated.

The representation is straightforward. To the end of each chromosome (string of bits interpretable as a point in the search space) we attach another bit string of the same length. Thus any string representation previously used with a traditional genetic algorithm can be employed in our scheme simply by doubling its length. The bits in the new section are interpreted as crossover punctuation, 1 for yes and 0 for no (i.e. 1=crossover, 0=no crossover). The loci in the punctuation (second) part of the string correspond one-to-one with the loci in the functional (first) part of the chromosome. Thus it is natural to think of these two parts of the chromosome as interleaved. Punctuation mark i tells whether crossover is or is not to occur at locus i. In figure 2, the punctuation marks are shown as "!" in the functional string.

It is common practice when beginning a genetic search to initiate a population of strings by randomly generating bits with equal probability for zero and one. We follow this practice for the functional string, but the probability of generating a one in the punctuation string is designated P_{2o} and is set externally. The influence of this variable was studied empirically.

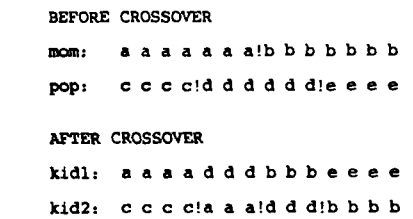


Figure 3. The action of punctuated crossover.

The mechanics of crossover governed by these punctuation marks is illustrated in figure 3. The bits from each parent string are copied one-by-one to one of the offspring from left to right. When a punctuation mark is encountered in either parent, the bits begin going to the other offspring (a crossover). When this happens, the punctuation marks themselves are also passed on to the offspring, just before the crossover takes effect. Thus we may think of the marks as being linked to the functional bit string to the left of its locus. A little experimentation with pencil and paper should serve to give the reader a sense of the redistribution possibilities of this process. Some parental distributions will result in all the punctuation marks being passed to one of the offspring and none to its sibling, while others will redistribute clumped distributions. When an offspring fails to survive the fitness-based selection step in its generation, its punctuation marks die with it. Thus, the dynamics of the distribution of these marks in the gene pool should reflect an accumulating experience about where is or is not a good place to crossover the genetic material in the pool.

The action of the mutation operator has traditionally been employed as a low level (i.e. small probability) defense against premature convergence of the gene pool. It seems consistent with our metaphor to allow the mutation operator equal access to the entire chromosome, both functional and punctuation parts. A few experiments supported this belief.

3. Empirical Evidence

We selected the task domain of function optimization (minimization) to test the capabilities of this new genetic algorithm and a set of five scalar-valued functions which has been used in the past to test genetic algorithms. These functions provide a range of characteristics of search problems and are summarized in table 1. Recently Grefenstette has found a configuration of the traditional genetic algorithm which performs consistently better on this function set than any previously known. [4]. We will use his results (which we will call BTGA for best traditional genetic algorithm) as a benchmark.

TABLE 1
Functions Comprising the Test Environment

Fcn	Dimensions	Space Size	Description
f1	3	1.0×10^{9}	parabola
f2	2	1.7×10^{6}	Rosenbrock's saddle
f3	5	1.0×10^{15}	step function
f4	30	1.0×10^{72}	quadratic with noise
f5	2	1.6×10^{10}	Shekel's foxholes

We adopted Grefenstette's strategy of setting a genetic algorithm to optimize a genetic algorithm (GA) in order to locate a good set of parameter values for our new procedure (a "meta-search"). The meta-level GA was a traditional GA that allowed vector-valued fitness (one dimension for each of the five functions) called VEGA [8]. The parameter set searched at this level included: population size, crossover rate, mutation rate, P_{so}, and scaling window. The performance measure was online average function value (i.e. an average of all trials in a run of 5000 function evaluations). For more details on these matters the reader is referred to Grefenstette's paper and its predecessors.

Since the performance of BTGA on each of the individual functions was not previously published, we first estimated this by running a BTGA on each one five times (n) with different random seeds. The results are given in table 2.

The results of the Meta-search revealed that the genetic algorithm with punctuated crossover (GAPC) performed well for the whole set of functions at the following parameter settings: population size = 40, crossover rate = 0.45, mutation rate = 0.002, P_{so} = 0.04, and scaling window = 3. Estimates of the performance of GAPC comparable to those for BTGA are given in table 3 along with results of two-tailed t-tests of the mean differences between them.

TABLE 2
Performance of Best Traditional Genetic Algorithm on Test Functions

Function	mean	s.d.	n	global optimum
f1	1.664	.1900	5	0
f2	25.16	4.497	5	0
f3	-27.78	.2111	5	-30
f4	24.28	1.383	5	0
f5	30.78	2.148	5	≈1

TABLE 3
Performance of Genetic Algorithm with Punctuated Crossover on Test Functions and a comparison with the Best Traditional Genetic Algorithm

Function	mean	s.d.	n	t-test	significance
f1	1.111	.1706	5	4.84	.01
f2	17.22	3.279	5	3.19	.05
f3	-27.90	.5907	5	0.42	ns
f4	20.32	1.320	5	4.63	.01
f5	14.81	1.475	5	13.71	.001

These results clearly show the superiority of GAPC. It statistically outperforms BTGA on four of the five functions and, is no worse on the other (f3). We believe the reason for this latter result lies in a floor effect. Both GAs find good solutions very quickly on f3 so that the online average rapidly approaches the global optimum. There is simply insufficient room for improvement to allow for a statistical difference. We believe the same explanation applies to the finding that no significant differences were noted between BTGA and GAPC when offline average was used as a criterion.

4. Characterization of GAPC Search

There are a number of questions about the realization of the performance characteristics we envisioned when this scheme was designed. Specifically, does the distribution of crossover marks adapt to the task environment in a meaningful way, is the process stable (i.e. does the number of punctuation marks in the population tend to vanish or saturate), and how many crossovers per mating does it settle upon (if it does settle)? In this section we present some evidence related to these questions that was collected while monitoring the searches reported above.

We define the population distribution of punctuation marks at time t as the sum of the punctuation bits at each locus across all the individuals in the population.

$$p_{so}(l,t) = \sum_{i=1}^{popsize} punct_i(l,t) \qquad (2)$$

The total number of punctuation marks in the population is then

$$T_{so}(t) = \sum_{l=1}^{L} p_{so}(l,t) \qquad (3)$$

Figure 4 shows a time history of $p_{so}(l,t)$ for 200 generations of one search of function f1. The x-axis is chromosome locus and the y-axis is both $p_{so}(l,t)$ and time

(generations). The vertical separation between successive generations is equal to popsize (i.e. the number of individuals in the population) so that, a locus at which every member of the population has a punctuation mark, will appear as a peak which just reaches the baseline of the line above. This figure shows that the initial distribution is flat since the random initialization of the punctuation marks does not favor one locus over any other. As time progresses, however, some loci tend to accumulate more punctuation marks than others. The location of these concentrations changes with time; a peak may appear and remain prominent for some generations only to die out as others emerge. The distribution of punctuation marks does indeed seem to adapt as the gene pool adapts.

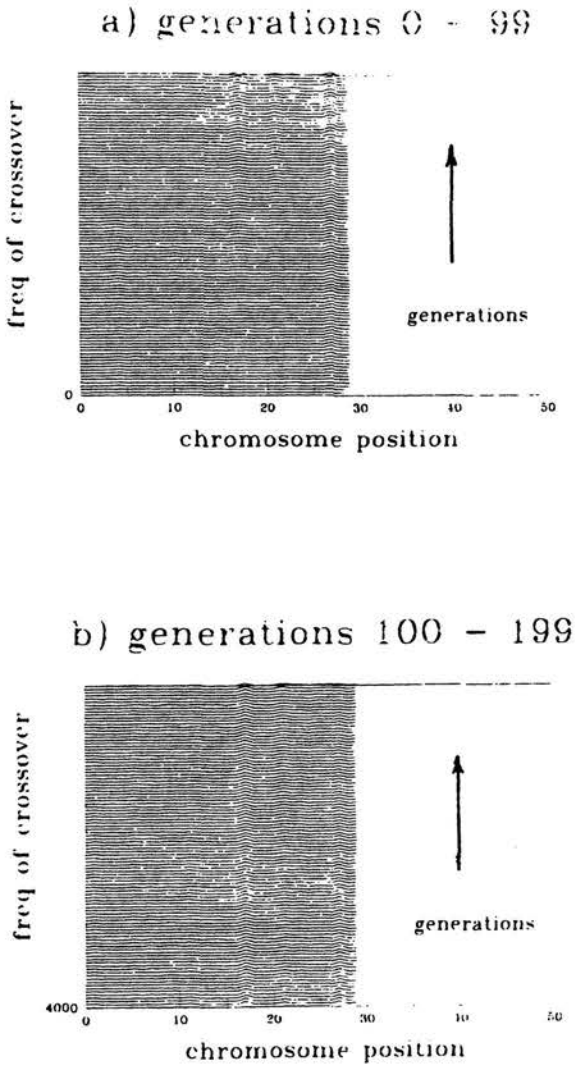


Figure 4. A time history of the distribution of punctuation marks for one run on f1.

Figure 5 shows a plot of $T_{20}(t)$ for the same run. Also shown are the $\max p_{20}(l,t)$ and $\min p_{20}(l,t)$. Although the total seems to be growing with no sign of stabilizing, the population is far from saturated†, the minimum remains zero until the last few generations and the maximum concentration at any one locus never exceeds 30 out of the possible 40 (i.e. popsize).

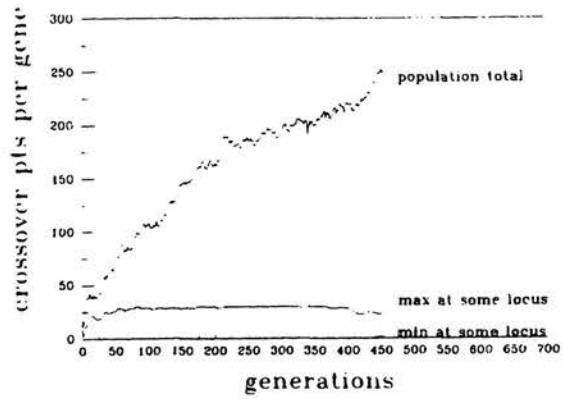


Figure 5. How the numbers of punctuation marks change with time. These data were from one search on f1, and were typical.

Figure 6 shows the average number of crossover events occurring per mating. This simple counting can be misleading, however, since the gene pool is converging as the generations progress. Note that when a crossover event swaps gene segments which are identical, it is an unproductive crossover event. When these events are discounted, we were surprised to see that the number of "productive" crossover events per mating remained nearly constant. What is more, the level at which this statistic holds appears to correlate strongly with L. See table 4. These results are not dissimilar to results reported by DeJong when experimenting with multiple crossover points [2]

TABLE 4
Time-averaged "Productive" Crossovers per Mating

Function	chromosome length	crossovers
f1	30	1.49
f2	24	0.87
f3	50	2.02
f4	240	8.64
f5	32	1.65

† Saturation would mean a punctuation bit at every one of the *popsize* $\times L$ ($40 \times 30 = 1200$) possible locations.

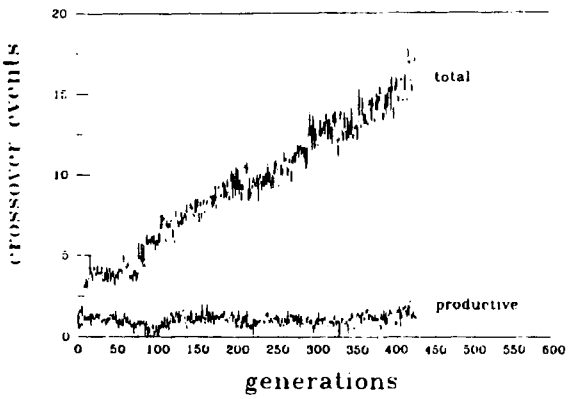


Figure 6. Total and "productive" crossover events per mating for one run on f1.

5. Conclusions

We have described a modified knowledge representation and crossover operator for use with genetic search. Its design was driven by intuition abstracted from Nature's mechanisms of crossover during meiosis. Experiments indicate that it performs as well or better than a traditional GA for a set of test problems, that exhibits a range of search space properties. Experiments on other test problems are continuing.

The distribution of crossover events evolves as the search progresses and the statistics of "productive" crossover events per mating indicate steady search effort even in the face of a converging gene pool. These statistics seem to correlate with chromosome length and are consistent with previous results.

We remain cautiously optimistic that continued experimentation will strengthen these conclusions and will lead to a robust approach to adaptive knowledge representation.

Acknowledgement

We wish to acknowledge the valuable contributions of D. Paul Benjamin to the conception of this project and to discussions of its implications.

References

1. B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts and J. D. Watson, *Molecular Biology of the Cell*, Garland Publishing, Inc., New York, 1983.
2. K. A. De Jong, Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, 1975.
3. K. A. De Jong, Adaptive System Design: A Genetic Approach, *IEEE Transactions on Systems, Man & Cybernetics SMC-10*,9 (September 1980), 566-574.
4. J. J. Grefenstette, Optimization of Control Parameters for Genetic Algorithms, *IEEE Transactions on Systems, Man & Cybernetics SMC-16*,1 (January-February 1986), 122-128.
5. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
6. J. H. Holland and J. S. Reitman, Cognitive Systems Based on Adaptive Algorithms, in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth (editor), Academic Press, New York, NY, 1978.
7. D. B. Lenat, The Role of Heuristics in Learning by Discovery: Three Case Studies, in *Machine Learning*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (editor), Tioga, Palo Alto, CA, 1983.
8. J. D. Schaffer, Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms, Ph.D. Thesis, Department of Electrical Engineering, Vanderbilt University, December 1984.
9. S. F. Smith, Flexible Learning of Problem Solving Heuristics Through Adaptive Search, *8th International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, August 1983.

GENETIC ALGORITHMS WITH SHARING FOR MULTIMODAL FUNCTION OPTIMIZATION

David E. Goldberg, The University of Alabama,
Tuscaloosa, AL 35487

and

Jon Richardson, The University of Tennessee
(formerly at The University of Alabama),
Knoxville, TN 37996

ABSTRACT

Many practical search and optimization problems require the investigation of multiple local optima. In this paper, the method of sharing functions is developed and investigated to permit the formation of stable subpopulations of different strings within a genetic algorithm (GA), thereby permitting the parallel investigation of many peaks. The theory and implementation of the method are investigated and two, one-dimensional test functions are considered. On a test function with five peaks of equal height, a GA without sharing loses strings at all but one peak; a GA with sharing maintains roughly equally sized subpopulations clustered about all five peaks. On a test function with five peaks of different sizes, a GA without sharing loses strings at all but the highest peak; a GA with sharing allocates decreasing numbers of strings to peaks of decreasing value as predicted by theory.

INTRODUCTION

Genetic algorithms (GAs) are finding increasing application in a variety of problems across a spectrum of disciplines (Goldberg & Thomas, 1986). This is so, because GAs place a minimum of requirements and restrictions on the user prior to engaging the search procedure. The user simply codes the problem as a finite length string, characterizes the objective (or objectives) as a black box, and turns the GA crank. The genetic algorithm then takes over, seeking near-optima primarily through the combined action of reproduction and crossover. These so-called simple GAs have proved useful in many problems despite their lack of sophisticated machinery and despite their total lack of knowledge of the problem they are solving. Yet as their usage has grown, several objections to their performance have arisen. Simple GAs have been criticized for sub-par performance on multimodal (multiply-peaked) functions. They have also been criticized for so-called premature convergence where substantial fixation occurs at most bit positions before obtaining sufficiently near-optimal points (Cavicchio, 1970; De Jong, 1975; Mauldin, 1984; Baker, 1985).

In this paper, we examine the first of these maladies and propose a cure borrowed from nature. In particular, our herbal remedy causes the formation of niche-like and species-like subdivision of the environment and population through the imposition of sharing functions. These sharing functions help mitigate unbridled head-to-head competition between widely disparate points in a search space. This reduction in competition between distant points thereby permits better performance on multimodal functions. As a side benefit we find that sharing helps maintain a more diverse population and more considered (and less premature) convergence. In the remainder of this paper, we review the problem and past efforts to solve it; we consider the theory of niche and speciation through Holland's modified two-armed bandit problem, and we compare the performance of a genetic algorithm both with and without the sharing function feature. Finally we examine extensions of the sharing function idea to permit its implementation in a wide array of problems.

MULTIMODAL OPTIMIZATION, GENETIC DRIFT, AND A SIMPLE GA

The difficulty posed by a multimodal problem for a simple genetic algorithm may be illustrated by a straightforward example. Figure 1 shows a bimodal function of a single variable: $f(x) = 4(x-0.5)^2$ coded by a normalized, five-bit binary integer. In this problem, the two optima are located at extreme ends of the one-dimensional space. If we start a genetic algorithm with a population chosen initially at random and let it run for a large number of generations, our fondest hope is that stable subpopulations cluster about the two optima (about 00000 and 11111). In fact if we perform this experiment, we find that the simple GA eventually clusters all of its points about one peak or the other.

Why does this happen? After all, doesn't the fundamental theorem of genetic algorithms (Holland, 1975; De Jong, 1975; Goldberg, 1986) tell us that exponentially increasing numbers of trials will be given to the observed best schemata? Yes it does, but the theorem assumes an infinitely large population size. In a finite size popula-

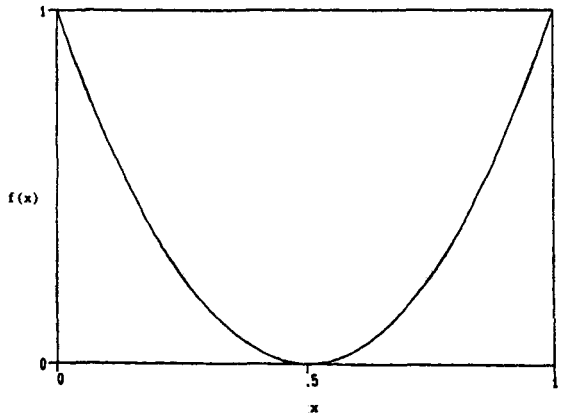


Figure 1. Bimodal function with equal peaks.

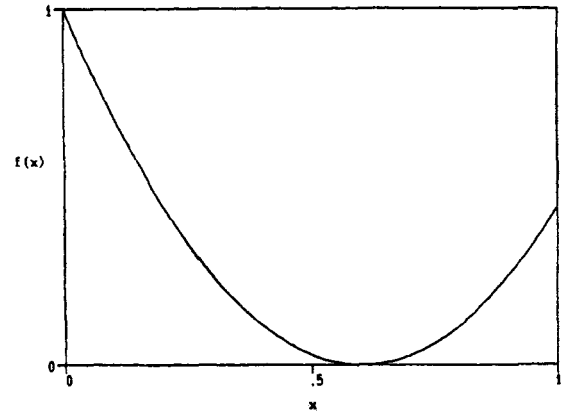


Figure 2. Bimodal function with unequal peaks.

tion, even when there is no selective advantage for either of two competing alternatives (as is the case for schemata 11*** and 00*** in the example problem) the population will converge to one alternative or the other in finite time (De Jong, 1975; Goldberg & Segrest, this volume). This problem of finite populations is so important that geneticists have given it a special name, genetic drift. Stochastic errors tend to accumulate, ultimately causing the population to converge to one alternative or another.

The convergence toward one optimum or another is clearly undesirable in the case of peaks of equal value. In multimodal problems where peaks of different altitudes exist, the desirability of convergence to the globally best peak is not so clear cut. In Figure 2 we see a bimodal function with unequal peaks: $f(x) = 2.8(x-0.6)^2$ with a five-bit normalized coding. If we are interested in obtaining only the global optimum, we should not mind the eventual convergence of the population to the leftmost point; however, this convergence is not always guaranteed. Small initial populations may allow sampling errors which overestimate the schemata of the rightmost points thereby permitting convergence to the wrong peak. Furthermore, in real world optimization we are often interested in having information about good, better, and best solutions. When this is so, it might be nice to see a form of convergence that permits stable subpopulations of points to cluster about both peaks according to peak fitness.

In either of these cases, we can argue for more controlled competition and less reckless convergence than is possible when we work with a simple, tripartite (reproduction, crossover, and mutation) genetic algorithm. For these reasons we turn to the theory of niche and speciation to find an appropriate model for naturally regulated competition.

THEORY OF SPECIES AND NICHE

The results of our initial gedanken-experimente (thought experiments) with simple GAs and multimodal functions are somewhat perplexing when juxtaposed with natural example. In our problem with equal peaks, the simple GA converges on one peak or the other even though both peaks are equally useful. By contrast, why doesn't nature converge to a single species? In our second problem with unequal peaks, we notice that the simple GA again converges to one peak, usually--but not always--the "correct" peak. How, when faced with a somewhat less fit species, does nature choose to limit population size before resorting to extinction? In both cases, nature has found a way to combat unbridled competition and permit the formation of stable subpopulations. In nature, different species don't go head to head. Instead they exploit separate niches (sets of environmental features) in which other organisms have little or no interest. In this section, we need to bridge the gap between natural example and genetic algorithm practice through the application of some useful theory.

Although there is a well-developed biological literature in both niche and speciation, its transfer to the arena of GA search has been limited. Like many other concepts and operators, the first theories directly applicable to artificial genetic search are due to Holland (1975). To illustrate niche and species, Holland introduces a modification of the two-armed bandit problem with distributed payoff and sharing. Let's examine his argument with a concrete formulation of the same problem.

Imagine a two-armed bandit as depicted in Figure 3. In the ordinary two-armed bandit problem (Holland, 1975; De Jong, 1975), we have two arms, a left arm and a right arm, and we have different payoffs associated with each arm. Suppose we have an expected payoff associated with the left arm of $25 and an expected payoff associated with the right arm of $75; in the standard two-armed bandit, we are unaware initially which arm pays the higher amount, and our dilemma is to min-

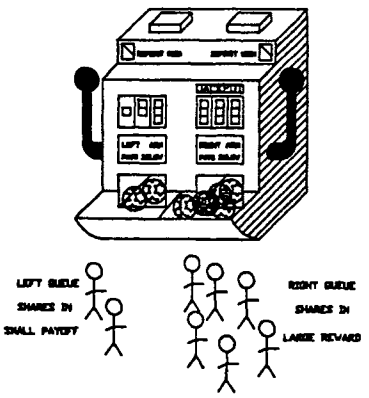


Figure 3. Sketch of the two-armed bandit with queues and sharing.

imize our expected losses over some number of trials. In this form, the two-armed bandit problem puts the tradeoffs between exploration and exploitation in sharp perspective. We can take extra time to experiment, but in so doing risk the possible gain from choosing the right arm, or we can experiment briefly and risk making an error once we choose the arm we think is best. In this way, the two-armed bandit has been used to justify the allocation strategy adopted by the reproductive plans of simple genetic algorithms.

This is not our purpose here. Instead we examine the modified two-armed bandit problem to put the concepts of niche and species in sharper focus. In the modified problem, we further suppose that we have a population of some number of players, say 100 players, and that each player may decide to play one arm or the other. If at this point we do nothing else, we simply create a parallel version of the original two-armed bandit problem where we expect that all players eventually line up behind the observed best (and actual best) arm. To produce the subdivision of species and niche, we introduce an important rule change. Instead of allowing a full measure of payoff for each individual, individuals who choose a particular arm are now forced to share the wealth derived from that arm with other players queued up at the arm. At first glance, this change appears to be quite minor. In fact, this single modification causes a strikingly and surprisingly different outcome in the modified two-armed bandit.

To see why and how the results change, we first recall that despite the different rules of the game, we still allocate population members according to payoff. In the modified game, an individual will receive a payoff which depends on the arm payoff value and the number of individuals queued up at that arm. In our concrete example, an individual lined up behind the right arm when all individuals are lined up behind that same arm receives an amount \$75/100 = \$0.75. On the other hand, an individual lined up behind the left arm when all individuals are queued there receives \$25/100 = \$0.25. In both cases, there is motivation for some individuals to shift lines. In the first case, a single individual changing lines stands to gain an amount \$25.00 - \$0.75 = \$24.25. The motivation to shift lines is even stronger in the second case. At some point in between we should expect there to be no further motivation to shift lines. This will occur when the individual payoffs are identical for both lines. If N is the population size, m_{right} and m_{left} are the number of individuals behind the right and left queues, and f_{right} and f_{left} are the expected payoff values from the right and left arms respectively, the equilibrium point may be calculated as follows:

$$\frac{f_{right}}{m_{right}} = \frac{f_{left}}{m_{left}}$$

In our example, this complete equalization of individual payoff occurs when 75 players select the right arm and 25 players select the left arm, because \$75/75 = \$25/25 = \$1.

This problem may be extended to the k-armed case directly, and the extension does not change the fundamental conclusions at all: the system attains equilibrium when the ratios of arm payoff to queue length are equal (Holland, 1975). The incorporation of forced sharing causes the formation of stable subpopulations (species) behind different arms (niches) in the problem. Furthermore, the number of individuals devoted to each niche is proportional to the expected niche payoff. This is exactly the type of solution we had hoped for when we considered the bimodal problems of Figures 1 and 2. Of course the extension of the sharing concept to real genetic algorithm search is more difficult than the idealized case implies. In a real genetic algorithm there are many, many arms and deciding who should share and how much should be shared becomes a non-trivial question. In the next section we will examine a number of current efforts to induce niche and species through indirect or direct sharing.

A BRIEF REVIEW OF CURRENT SCHEMES

A number of methods have been implemented to induce niche and species in genetic algorithms. In some of these techniques the sharing comes about indirectly. Although the two-armed bandit problem is a nice, simple abstract model of niche and species formation and maintenance, nature is not so direct in divvying up her bounty. In natural settings, sharing comes about through crowding and conflict. When a habitat becomes fairly full of a particular organism, individuals are forced to share available resources.

Cavicchio's (1971) dissertation study was one of the first to attempt to induce niche-like and species-like behavior in genetic algorithm search. Specifically, he introduced a mechanism he called

preselection. In this scheme, an offspring replaces the inferior parent if the offspring's fitness exceeds that of the inferior parent. In this way diversity is maintained in the population because strings tended to replace strings similar to themselves (one of their parents). Cavicchio claimed to maintain more diverse populations in a number of simulations with relatively small population sizes (n=20).

De Jong (1975) has generalized preselection in his crowding scheme. In De Jong crowding, individuals replace existing strings according to their similarity with other strings in an overlapping population. Specifically, an individual is compared to each string in a randomly drawn subpopulation of CF (crowding factor) members. The individual with the highest similarity (on the basis of bit-by-bit similarity count) is replaced by the new string. Early in the simulation, this amounts to random selection of replacements because all individuals are likely to be equally dissimilar. As the simulation progresses and more and more individuals in the population are similar to one another (one or more species have gotten a substantial foothold in the population) the replacements of individuals by similar individuals tends to maintain diversity within the population and reserve room for one or more species. De Jong has had success with the crowding scheme on multimodal functions when he used crowding factors CF=2 and CF=3. De Jong's crowding has subsequently been used in a machine learning application (Goldberg, 1983).

Booker (1982) discusses a direct application of the sharing idea in a machine learning application with genetics-based, classifier systems. In classifier systems, a sub-goal reward mechanism called a bucket brigade passes reward through a network of rules like money passing through an economy. Booker suggests that appropriately sized subpopulations of rules can form in such systems if related rules are forced to share payments. This idea is sound and has been forcefully demonstrated in Wilson's recent work with boolean function learning (Wilson, 1986); however, it does not transfer well to function optimization, because unlike classifier systems, there is no general way in function optimization to determine which strings are related.

Shaffer (1984) has used separate, fixed size subpopulations in his study of vector evaluated genetic algorithms (VEGA). In this study, each component of the vector (each criterion or objective measure) is mapped to its own subpopulation where separate reproduction processes are carried out. The method has worked well in a number of trial functions; however, Shaffer has expressed some concern over the procedure's ability to handle middling nondominated individuals--individuals that may be Pareto optimal but are not extremal (or even near extremal) along any single dimension. Furthermore, although the study does use separate subpopulations, it is unclear how the same method might be applied to the more usual single-criterion optimization problem.

A direct exploration of biological niche theory in the context of genetic algorithms is contained in Perry's (1985) dissertation. In this work, Perry defines a genotype-to-phenotype mapping, a multiple-resource environment, and a special entity called an external schema. External schemata are special similarity templates defined by the simulation designer to characterize species membership. Unfortunately, the required intervention of an outside agent limits the practical use of this technique in artificial genetic search. Nonetheless, the reader interested in the connections between biological niche theory and GAs may be interested in this work.

Grosso (1985) also maintains a biological orientation in his study of explicit subpopulation formation and migration operators. Multiplicative, heterotic (problems with diploid structures where a heterozygote is more highly fit than the homozygote) objective functions are used in this study, and as such, the results are not directly applicable to most artificial genetic search; however, Grosso was able to show the advantage of intermediate migration rate values over either isolated subpopulations (no migration) and panmictic (completely mixed) subpopulations. This study suggests that the imposition of a geography within artificial genetic search may be another useful way of assisting the forming diverse subpopulations. Further studies are needed to determine how to do this in more general artificial genetic search applications.

Although he has not directly addressed niche and species, Mauldin (1984) has attempted to better maintain diversity in genetic algorithms through his uniqueness operator. The uniqueness operator arbitrarily returns diversity to a population whenever it is judged to be lacking. To implement uniqueness, Mauldin defines a uniqueness parameter k_u that may decrease with time (similar to the cooling of simulated annealing). He then requires that for insertion in a population, an offspring must be different than every population member at a minimum of k_u loci. If the offspring is not sufficiently different, it is mutated until it is. By itself, uniqueness is little more than a somewhat knowledgeable (albeit expensive) mutation operator. That it is useful in improving offline (convergence) performance is not unexpected. Grefenstette (1986) has recently supported the notion of fairly high mutation probabilities (p_m = 0.01 to 0.1) when convergence to the best is the main goal. It is interesting to note that uniqueness combined with De Jong's crowding scheme worked better than either operator by itself (Mauldin, 1984). This result suggests that maintaining diversity for its own sake is not the issue. Instead, we need to maintain appropriate diversity--diversity that in some way helps cause (or has helped cause) good strings. In the next section, we show how we can maintain appropriate diversity through the use of sharing functions.

SHARING FUNCTIONS

In attempting to induce species, we must either directly or indirectly cause intraspecies sharing, but we are faced with two important questions: who should share, and how much should be shared? In natural systems, these two questions are answered implicitly through conflict for finite resources. Different species find different combinations of environmental factors--different niches--which are relatively uninteresting to other species. Individuals of the same species use those resources until there is conflict. At that point, they vie for the same turf, food, and other environmental resources, and the increased competition and conflict cause individuals of the same species to share with one another, not out of altruism, but because the resources they give up are not worth the cost of the fight. It might be possible to induce similar conflict for resources in genetic optimization. Unfortunately, in many optimization problems, there is no natural definition of a resource. As a result, we must invent some way of imposing niche and speciation on strings based on some measure of their distance from each other. We do just this with what we have called a sharing function.

A sharing function is nothing more than a way of determining the degradation of an individual's payoff due to a neighbor at some distance as measured in some similarity space. Mathematically, we introduce a convenient metric d over our decoded parameters x_i (the decoded parameters are themselves functions of the strings $x_i = x_i(s_i)$):

$$d_{ij} = d(x_i, x_j)$$

Alternatively, we may introduce a metric over the strings directly:

$$d_{ij} = d(s_i, s_j)$$

In this paper, we use a metric defined over the decoded parameters x_i (phenotypic sharing); later on, we briefly consider the use of metrics defined over the strings (genotypic sharing). However we choose a metric, we define a sharing function sh as a function of the metric value sh = sh(d) with the following three properties:

1. $0 \le sh(d) \le 1$, for all d
2. $sh(0) = 1$
3. $\lim_{d \to \infty} sh(d) = 0$

Many sharing functions are possible. Power law functions are convenient:

$$sh(d) = 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha}, \quad d < \sigma_{share}$$
$$= 0, \quad \text{otherwise}$$

In this equation, σ_{share} and α are constants, and Figure 4 displays power law sharing functions with α values equal to one, greater than one, and less than one.

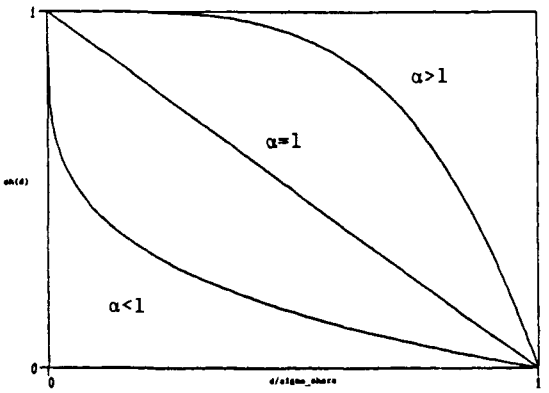


Figure 4. Power law sharing functions sh=sh(d).

Once we have selected a metric and a sharing function, it is a simple matter to determine a string's realized or shared fitness. We define a string's shared fitness f' as its potential fitness divided by its niche count m_i':

$$f_i' = \frac{f_i}{m_i'}$$

The niche count m_i' for a particular string i is taken as the sum of all share function values taken over the entire population:

$$m_i' = \sum_{j=1}^{N} sh(d_{ij}) = \sum_{j=1}^{N} sh(d(x_i, x_j))$$

Note that the sum includes the string itself. Thus, if a string is all by itself in its own niche ($m_i' = 1$), it receives its full potential fitness value. Otherwise the sharing function derates fitness according to the number and closeness of neighboring points.

RESULTS

We evaluate the use of sharing functions through computational experiments on two multimodal problems. The first function is a periodic function with five peaks of equal magnitude:

$$f_1(x) = \sin^6(5.1\pi x+.5)$$

We compare the performance of a simple genetic algorithm with stochastic remainder selection, crossover, no mutation, and no sharing to the performance of the same GA with sharing. We use a triangular sharing function (α=1) with σ_{share} = 0.1; the metric d is taken as the absolute value of the difference between the string x values. Length 30 binary strings are decoded as unsigned binary integers and normalized by the constant 2^{30}-1. Genetic algorithm parameters have been held constant across all runs as follows:

Probability of mutation p_m = 0.0

Probability of crossover p_c = 0.8

Population size = 50

Maximum number of generations = 100

We seek methods that maintain appropriate diversity without introducing arbitrary diversity through mutation or other means. Therefore, we have set the mutation probability to zero to put the sharing function technique to its most stringent test: if appropriate diversity can be maintained without mutation, we should expected similar or better (off-line) performance when mutation is present.

Runs with and without sharing are started from the same population generated uniformly at random. After 50 generations the run without sharing has lost all points at two peaks as shown in Figure 5. By contrast, the run with sharing has stable subpopulations roughly equal in size at all five peaks as shown in Figure 6. Similar graphs at generation 100 are shown in Figures 7 and 8. Note how the run without sharing (Figure 7) has completely converged to a single peak even though there is no selective advantage for any peak. By contrast, the run with sharing remains committed to stable subpopulations clustered about each peak. This latter result is especially remarkable considering that no mutation has been used. With no mutation, once an allele is lost at a particular locus, there is no way to get it back. The existence of stable subpopulations about each peak shows how the sharing function maintains appropriate diversity--the necessary, sometimes competing building blocks--required to exploit all five peaks.

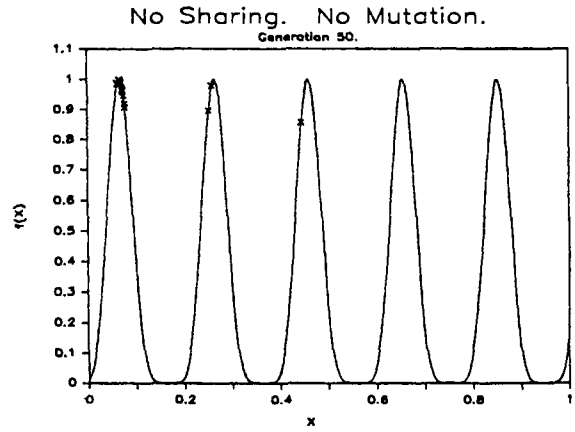


Figure 5. GA without sharing concentrates points at three peaks after 50 generations on function f_1 (equal peaks).

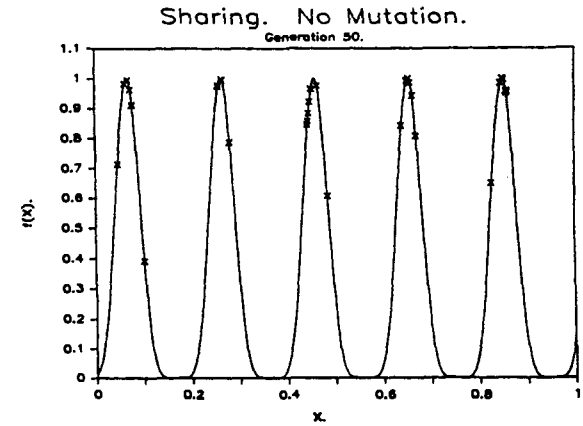


Figure 6. GA with sharing distributes points to all five peaks after 50 generations on function f_1 (equal peaks).

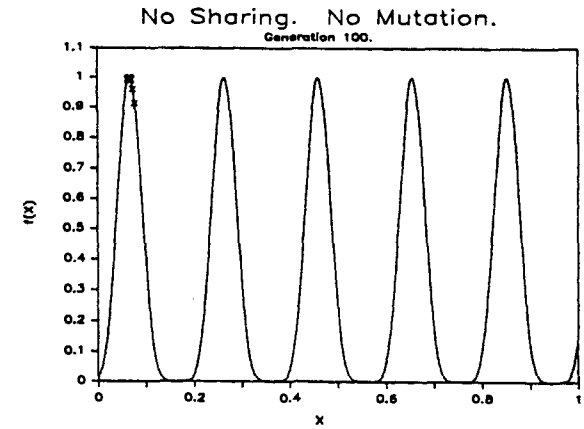


Figure 7. GA without sharing concentrates all points on a single peak after 100 generations on function f_1 (equal peaks).

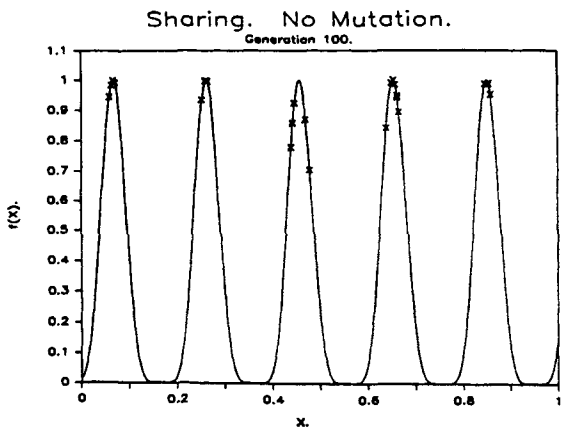


Figure 8. GA with sharing continues to distribute points among all five peaks after 100 generations on function f_1 (equal peaks).

The second test function $f_2(x)$ has five peaks with decreasing peak magnitude as given by the following equation:

$$f_2(x) = f_1(x) \cdot e^{\left[-4\ln 2 \frac{(x-0.0667)^2}{0.8^2} \right]}$$

We make comparisons of the two GAs, with and without sharing, using the same parameters and string coding as before. Specifically, we compare the two cases at generation 100. By that time, the genetic algorithm without sharing has allocated all of its trials to the highest peak as shown in Figure 9. By contrast, the GA with sharing forms stable clusters of points about four of the five highest peaks with cluster size roughly proportional to peak fitness as shown in Figure 10. This is the kind of performance we predicted earlier, except for the lack of strings at the lowest peak. To understand why this has occurred we briefly return to the theory of sharing presented earlier.

From our earlier discussion, we expect a stable equilibrium to form when the following equations hold true:

$$\frac{f_i}{m_i'} = \frac{f_{i+1}}{m_{i+1}'}$$

for all niches i, and

$$\sum_{i=1}^{M} m_i' = N$$

where M is the number of niches and N is the population size. It may be shown that this set of equations predicts proportions of niche members as the ratio of niche fitness to the total fitness. On function f_2, we expect a total number of trials to be allocated to the lowest peak as follows:

$$50*0.075/(0.075+0.22+0.5+0.85+1.0) = 1.42$$

In other words, we expect approximately one individual to remain at that peak, and we should not be surprised that no strings cluster there. If we want to maintain a subpopulation at such a low peak, our theory suggests that we need a larger population to overcome the unavoidable errors of stochastic sampling and selection.

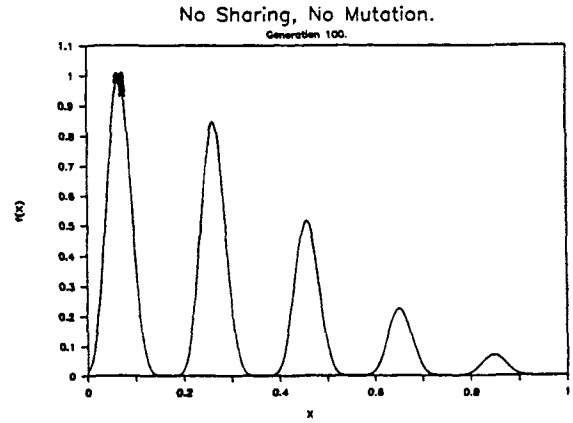


Figure 9. GA without sharing concentrates all points on highest peak (at generation 100) on function f_2 (decreasing peaks).

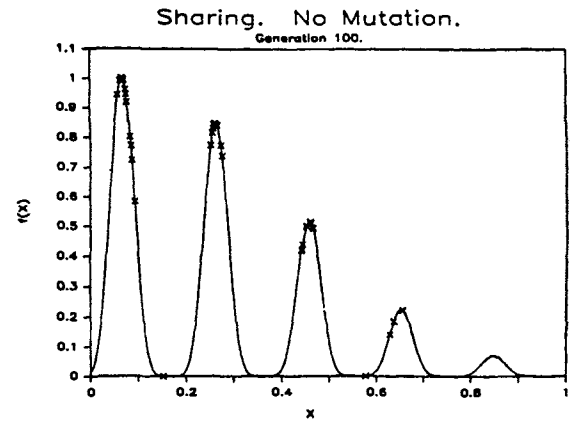


Figure 10. GA with sharing distributes points to all but lowest peak (at generation 100) on function f_2 (decreasing peaks). Lowest peak has expected population size of only one member, not enough to overcome selection and sampling errors.

EXTENSIONS

The method of sharing functions is not limited to one-dimensional problems. Sharing functions may be evaluated using any reasonable metric that includes any number of problem parameters. Additionally, there is no reason to limit the method to phenotypic sharing (where the measures are calculated based on differences in the phenotype--the decoded problem parameters). As an alternative, sharing functions may be evaluated using the genotype (the string) directly. In this form, the Hamming distance (the number of different bits) may be an especially useful metric.

In many cases, there may be no need to perform the sharing calculations as precisely as has been implied by the above equations. With the full formulation, $\binom{N}{2}$ sharing function evaluations are required (N^2 if symmetry is not exploited) to calculate the niche count values m_i exactly. This level of computation may be reduced by taking a random sample of k (k<<N) share function values and extrapolating the mean. If the mean of the k share function evaluations for string i is μ_i and the population is of size N, then the following formula provides a reasonable way to estimate the niche count m_i':

$$m_i' = (N-1)\mu_i + 1$$

Although this approximate niche count method has not been tested, Monte Carlo sampling techniques have been adopted in at least one other GA study with success (Grefenstette & Fitzpatrick, 1985). There is every reason to suspect that cheaper, approximate niche count estimates may be used without excessive performance degradation.

CONCLUSION

In this paper, we have developed a method for improving the performance of genetic algorithms in multimodal function optimization problems. This method uses sharing functions to induce artificial analogs to the natural concepts of niche and species, thereby permitting the formation of stable, non-competing subpopulations of points surrounding important peaks in the search space. The method has been tested on two multimodal functions, one with peaks of equal size and one with peaks of decreasing size. In both cases, the genetic algorithm with sharing is able to maintain stable subpopulations about significant peaks while an identical GA without sharing is unable to maintain points at more than a single peak. Additionally, the GA with sharing is also able to maintain stable subpopulations of appropriate size: the number of points in each cluster is roughly proportional to the peak fitness value. This automatic allocation of resources in a reasonable fashion should not only be transferable to other multimodal optimization problems, it should help in maintaining appropriate diversity in genetic algorithms without resorting to mutation and mutation-like operators that unnecessarily degrade on-line performance. These proof-of-principle results should permit the extension of these methods to other larger and more complex problems of genetic optimization.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant MSM-8451610.

REFERENCES

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In J. J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and Their Applications (pp. 101-111). Pittsburgh: Carnegie-Mellon University.

Booker, L. B. (1982). Intelligent behavior as an adaptation to the task environment. (Doctoral dissertation, Technical Report No. 243. Ann Arbor: University of Michigan, Logic of Computers Group). Dissertations Abstracts International, 43(2), 469B. (University Microfilms No. 8214966)

Cavicchio, D. J. (1970). Adaptive search using simulated evolution. Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 36(10), 5140B. (University Microfilms No. 76-9381)

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 44(10), 3174B. (University Microfilms No. 8402282)

Goldberg, D. E. (1986). Simple genetic algorithms and the minimal deceptive problem (TCGA Report No. 86003). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., & Thomas, A. L. (1986). Genetic algorithms: A bibliography (TCGA Report No. 86001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, SMC-16(1), 122-128.

Grefenstette, J. J., & Fitzpatrick, J. M. (1985). Genetic search with approximate function evaluations. In J. J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and Their Applications (pp. 112-120). Pittsburgh: Carnegie-Mellon University.

Grosso, P. B. (1985). Computer simulation of genetic adaptation: Parallel subcomponent interaction in a multilocus model. (Doctoral dissertation, University of Michigan, University Microfilms No. 8520908).

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

Mauldin, M. L. (1984). Maintaining diversity in genetic search. Proceedings of the National Conference on Artificial Intelligence, 247-250.

Perry, Z. A. (1984). Experimental study of speciation in ecological niche theory using genetic algorithms. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 45(12), 3870B. (University Microfilms No. 8502912)

Shaffer, J. D. (1984). Some experiments in machine learning using vector evaluated genetic algorithms. Unpublished doctoral dissertation, Vanderbilt University, Nashville.

Wilson, S. W. (1986). Classifier system learning of a boolean function (Research Memo RIS-27r). Cambridge: Rowland Institute for Science.