Punctuated Equilibria: A Parallel Genetic Algorithm

J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. Richards
Department of Computer Science
University of Virginia

Computer Science Report No. TR–87-05
Revised May 31, 1987

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 ''trajectory'' 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 the process is 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 the ''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 study of parallel 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 algorithms. We assume that the interconnection network is sparse, 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 the GA approach. In order to probabilistically select parents for the crossover operation, 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_{ij}. We need to find an mapping p, where object i is assigned to position $p(i)$, that minimizes the objective function

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

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 with the ''gene flow'' of the 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 $n \times N$. 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 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, $S = |S_{ij}|$, 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

```
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.

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 without replacement; the good solutions will only ''propagate'' at the beginning of the next epoch. (The selection of each S_{ij} is not done probabilistically. 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*, $0 \leq M \leq 1$, 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 $1, 2, \ldots, m$,

```
for $G$ iterations do
    for $n \times C$ iterations do
        select two solutions
        crossover those solutions
        add offspring to subpopulation
    endfor
    calculate fitnesses
    select a population of $n$ elements
    generate $n \times M$ random mutations
endfor
```

Figure 2 — The genetic algorithm used within an epoch at each processor.

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^{th} position. Briefly, their crossover copied a contiguous portion of one parent into the offspring, while having the other parent copy over as many other positions as possible. Smith [SMIT85] 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] with 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 ''half'' 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. We 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 [COHO87, WHIT84]. Therefore we used

$$fitness(x) = \frac{(\mu_s - score(x)) + \alpha\sigma}{2\alpha\sigma} \tag{2}$$

where μ_s is the mean of the scores, σ is the s.d., and α is 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.

Our fitness measure has several advantages. It is somewhat problem independent, so that 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 $N \times E \times G$ 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 evolutionary time line, the parallel algorithm has multiple, interrelated evolutionary time lines. The ''interrelated'' qualification is quite important because the parallel system does more than just create

N divergent time lines. As with 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 $N \times E \times G \times n$ to be an indicator of the total number of solutions 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 length of the mutation block.

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); $\bar{s}$ is the average of the best OLA score from each example with the specified settings; and $\hat{s}$ 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.

Results For Three Problem Instances

I	C	M	S	μ	n	α	N	E	G	$\hat{s}$	$\bar{s}$	s^*
1	0.50	0.05	15	3	80	3.00	4	2	50	9600	9600	9600
							1	1	400	9600	9600	
							4	1	100	9600	9600	
2	0.50	0.05	15	3	80	3.00	4	6	50	19200	19200	19200
							1	1	1200	19200	19200	
							4	1	300	19200	21757	
3	0.50	0.05	15	3	80	3.00	4	6	50	15510	15551	15450
							1	1	1200	15540	15622	
							4	1	300	15540	15596	

Table 1

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 to one, i.e., one population creating G generations. For a comparable uniprocessors, this 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 $N \times E \times G \times n$ 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.

The first type of problem instance, with unique optima, has a cost matrix of the following form:

$$C_1(9) = \begin{bmatrix} 0 & A & B & C & D & 0 & 0 & 0 & 0 \\ A & 0 & A & B & C & D & 0 & 0 & 0 \\ B & A & 0 & A & B & C & D & 0 & 0 \\ C & B & A & 0 & A & B & C & D & 0 \\ D & C & B & A & 0 & A & B & C & D \\ 0 & D & C & B & A & 0 & A & B & C \\ 0 & 0 & D & C & B & A & 0 & A & B \\ 0 & 0 & 0 & D & C & B & A & 0 & A \\ 0 & 0 & 0 & 0 & D & C & B & A & 0 \end{bmatrix} .$$

where $m = 9$ and $A \gg B \gg C \gg D \geq 0$. We believe the solution spaces for such problems instances to be "convex" in some sense, and therefore "easy." Instance one ($I = 1$) 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 $\overline{s} = s^*$.

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:

$$C_2(18) = \begin{bmatrix} C_1(9) & \mathbf{0} \\ \mathbf{0} & C_1(9) \end{bmatrix} .$$

Note that two groups of 9 are uncoupled and that this is tantamount to solving two disjoint problems. A, B, C, and D in $C_2(18)$ were as above. The settings and results for this cost matrix are shown as instance two ($I = 2$) 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

$$C_3(9) = \begin{bmatrix} 0 & A & A & C & C & C & C & C & C \\ A & 0 & A & C & C & C & C & C & C \\ A & A & 0 & B & C & C & C & C & C \\ C & C & B & 0 & A & A & C & C & C \\ C & C & C & A & 0 & A & C & C & C \\ C & C & C & A & A & 0 & B & C & C \\ C & C & C & C & C & B & 0 & A & A \\ C & C & C & C & C & C & A & 0 & A \\ C & C & C & C & C & C & A & A & 0 \end{bmatrix} .$$

We assume $A > B > C$. The intra-block cost, A, causes primary clustering, and the inter-block cost for adjacent 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 objects comprised of six blocks, and we let $A = 50$, $B = 30$, and $C = 15$. The results are shown as instance three ($I = 3$) in Table 1.

The results shown in Table 2 are presented to indicate the effects of changing individual design variables. The problem instance used $C_3(18)$ with $A = 40$, $B = 30$, and $C = 15$. Note the slight reduction in A was intended to make the optimum solution more elusive. The settings for the base case are shown in the first row. In the remaining rows we show the altered value of a particular variable and the obtained results. Again, the averages were taken over four example runs. In terms of $\bar{s}$, the most dramatic changes were due to reducing α, and due to increasing E. The effect of decreasing α is to make the fitness measure more sensitive to smaller differences between solutions that are near the current best for the subpopulation. In this way incremental improvements are given more of an opportunity to survive and create further improvements.

The increase in E effectively provides more communication opportunities between the subpopulations. The last row of Table 2 and instance three from Table 1 provide the strongest

The Effects Of Changing The Design Variables											
C	M	S	μ	n	α	N	E	G	$\hat{s}$	$\bar{s}$	s^*
0.50	0.05	15	3	80	3.00	4	6	50	15225	15318	
0.80									15240	15315	
	0.20								15270	15311	
		40							15240	15343	
			12						15270	15360	15210
				50					15225	15371	
					1.50				15210	15258	
					0.50				15210	15225	
							12	25	15210	15266	

Table 2

experimental results to date for the effectiveness of the parallel genetic algorithm with punctuated equilibria. These promising effects prompted an experiment combining the modifications in the last two rows of Table 2. We obtained the optimal solution in 5 out of 6 runs.

CONCLUSIONS AND EXTENSIONS

In attempting to develop parallel algorithms one always wants to obtain the simple speed-up of having more processors to do more instructions in the same ''wall clock'' time. However, there is evidence that in attempting to develop parallel versions of previously known algorithms one often derives a modified formulation which comprises more fundamental efficiencies. We believe this to be the case for our parallel genetic algorithm with punctuated equilibria. The partition into subpopulations specifies a simple and balanced mapping of the workload to a non-shared memory, multiprocessor system, while the intra-epoch isolation and inter-epoch communication provide a fundamental modification to the basic genetic algorithm that will generate better solutions while considering a smaller total number of individuals.

Several extensions to the model have been considered and are being attempted. For example, the use of fixed-size subpopulations is not suggested by the natural evolutionary setting. When a processor receives a surplus of very fit solutions it makes sense to retain most of them. However it is clear there should be some mechanism for limiting the size of each subpopulation as well as the total size. Varying-sized groups will create data management and coordination problems, and it is not clear they are worth the additional computational load.

Our model is essentially synchronous, though it is easily realized asynchronously with handshaking. A truly asynchronous model would allow each processor to decide for itself whether it has reached equilibrium and should begin another epoch. At that point it could poll its neighbors, asking for subsets of their current subpopulations to be sent to it. Global termination, while somewhat arbitrary before, becomes even more difficult.

ACKNOWLEDGEMENTS

The authors' work has been supported in part by the Jet Propulsion Laboratory of the California Institute of Technology under Contract 957721 to the University of Virginia. The work of James Cohoon has been supported additionally in part by the National Science Foundation through grant DMC 8505354. Their support is greatly appreciated.

REFERENCES

[BETH81] A. Bethke, *Genetic Algorithms as Function Optimizers, Ph.D. Thesis*, Department of Computer and Communication Sciences, University of Michigan, 1981.

[COHO86] J. P. Cohoon and W. D. Paris, Genetic Placement, *IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, 1986, 422-425.

[COHO87] J. P. Cohoon and M. T. Roberson, *Jump Starting Simulated Annealing*, Department of Computer Science, University of Virginia, 1987.

[DAVI85] L. Davis, Job Shop Scheduling with Genetic Algorithms, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, 136-140.

[ELDR72] N. Eldredge and S. J. Gould, Punctuated Equilibria: An Alternative to Phyletic Gradualism, in *Models of Paleobiology*, T. J. M. Schopf (ed.), Freeman, Cooper and Co., 1972, 82-115.

[ELDR85] N. Eldredge, *Time Frames*, Simon and Schuster, 1985.

[FOUR85] M. P. Fourman, Compaction of Symbolic Layout Using Genetic Algorithms, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, 141-150.

[GARE79] M. R. Garey and D. S. Johnson, *Computers And Intractability - A Guide To The Theory Of NP-Completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[GOLD83] D. E. Goldberg, *Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Learning Rules, Ph.D. Thesis*, Department of Civil Engineering, University of Michigan, 1983.

[GOLD85] D. E. Goldberg and R. Lingle, Jr., Alleles, Loci, and the Traveling Salesperson Problem, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, 154-159.

[GREF85] J. J. Grefenstette, R. Gopal, B. J. Rosmaita and D. Van Gucht, Genetic Algorithms for the Traveling Salesperson Problem, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, 160-168.

[HOLL75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[KIRK83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, *Science 220*, 4598 (May 13, 1983), 671-680.

[SMIT85] D. Smith, Bin Packing with Adaptive Search, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, 202-206.

[WHIT84] S. R. White, Concepts of Scale in Simulated Annealing, *International Conference on Computer Design: VLSI in Computers Proceedings*, Port Chester, NY, 1984, 646-651.

[WRIG32] S. Wright, The Roles of Mutation, Inbreeding, Crossbreeding, and Selection in Evolution, *Proceedings of the Sixth International Congress of Genetics 1*, (1932), 356-366.