Complex Systems **5** (1991) 443–458

Improved Evolutionary Optimization of Difficult Landscapes: Control of Premature Convergence through Scheduled Sharing

Michael E. Palmer*
Department of Computer Science, California Institute of Technology, Pasadena, CA 91125, USA

Stephen J. Smith†
Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, USA

Abstract. Massively parallel computers afford to genetic algorithms the use of very large populations, which allow the algorithms to attack more difficult optimization problems than were feasible in the past. Optimization performance on difficult search spaces—those that are both vast and have large numbers of local optima—can be particularly crippled by a common problem of genetic algorithms: premature convergence of the population to a suboptimum. In nature, however, competition between like organisms prevents total convergence, and this competition effect can also be introduced to the standard genetic algorithm by an extension called *sharing*. Sharing forces organisms within a *niche* to compete. In past work, the size of the niche has been fixed, and calculated by hand for simple test problems.

This paper introduces an improvement to fixed-niche sharing called *scheduled sharing*, which (1) allows for the application of sharing to complex problems where there is no definable best niche size, and (2) does not violate the "black box" principle in the calculation of niche size, instead borrowing from simulated annealing an exponentially decreasing schedule. We show that scheduled sharing inhibits convergence and improves performance for optimization problems that are difficult relative to the size of the population used.

*Electronic mail address: `mep@vlsi.cs.caltech.edu`
†Electronic mail address: `smith@think.com`

1. Introduction

The large populations afforded to genetic algorithms by massively parallel computers—currently many thousands of organisms—can be applied to the optimization of more difficult spaces than have been previously attempted. When searching difficult spaces—spaces that are both vast and have many local optima—genetic algorithms are particularly susceptible to the problem of premature convergence to local optima. Through various mechanisms, many optimization algorithms can become "trapped" in local optima of a space before they have found the global optimum. In genetic algorithms, this occurs when one type of organism multiplies too quickly before the global optimum has been found, destroying variation in the population, which is needed for a broader search of the space.

This paper introduces an extension to the standard genetic algorithm called *scheduled sharing*, and studies its effects on large-population genetic optimization of rugged two-dimensional landscapes. Scheduled sharing attacks the problem of premature convergence to improve offline performance by requiring organisms within a *niche* to compete with their neighbors. The niche size is initially large, encouraging the organisms to spread evenly across the space; it then decays exponentially, allowing gradual clustering of organisms in areas of greater interest.

Convergence in genetic algorithms and in nature

Genetic algorithms exploit past experience by allowing the propagation of organisms that have had past success. In addition, crossover allows the evolution of successful *schemata*—segments of genetic material smaller than the entire genotype. Natural selection provides the mechanism for the propagation of successful organisms, but also tends to push a population toward domination by many copies of a single, relatively fit genotype. In the standard genetic algorithm (DeJong's R4 algorithm [6]), there is no force specifically working against this domination, and the fate of most populations, after several generations, is domination by many copies of nearly identical organisms. Once this total convergence to a single solution has occurred, crossover can no longer generate variation in the population; remaining is mutation, a slower, blind source of variation. After total convergence, the search for better solutions is effectively stopped. In the case that the global optimum is found before convergence, the algorithm is considered to have performed as was intended; however, if the global optimum, or a point close enough to it, was not found, then the convergence is called *premature*. Sometimes early convergence saves time, and is sufficient to find a good solution; but sometimes, especially in realistic and difficult spaces, early convergence dismisses potentially rewarding search areas before they have been probed adequately.

There exist extensions to the standard genetic algorithm that specifically address the problem of premature convergence. For instance, see Shaefer's ARGOT [12]; Booker's work on improving the crossover operator [4];

Mauldin's uniqueness requirement [10]; and Baker's "percent involvement" method to predict rapid convergence and respond to it [2].

This paper is mainly concerned with another such extension to the standard algorithm called *sharing*, as described in [7, 5], which makes an analogy to natural competition between like organisms. In nature, the population of a single type of organism does not continue to increase without limit. As one type of organism multiplies to a large subpopulation, this subpopulation consumes more of certain resources present in the environment, and these resources become scarce. Similar individuals must then compete with each other for the same scarce resources, causing them all to suffer from want. Growth of this subpopulation is then slowed, and finally settles around some "carrying capacity," at which the number of organisms of a particular type is in equilibrium with the availability of the resources that they consume.

Simulation of the *sharing* effect can be added to the standard genetic algorithm by supposing abstract resources that all organisms exploiting a particular *niche* must compete for, or share. If there are too many organisms within a niche, the shortage of these resources decreases the fitness of all the organisms within the niche. It is assumed that two organisms that are adjacent in the search space will consume more of the same resources than two organisms that are far apart in the search space. (Note that the resources are never explicitly simulated but are simply an analogy to population biology.) The effect is introduced by multiplying an organism's phenotypic fitness by a *sharing factor*, a number between zero and one that is inversely proportional to the number of neighbors the organism has near it within a distance called the *niche size*. Distance is measured by some metric that is defined on the phenotype or genotype space. For instance, in this work, our algorithms searched for maxima on rugged two-dimensional landscapes: the phenotypes were (x, y) pairs. We therefore used a simple two-dimensional cartesian distance function. Alternatively, distance could be measured by hamming distance between genotypes.

Fixed-niche sharing as described in [7, 5] requires the *tuning* of the size of the niche to a particular search function. For instance, when searching a one-dimensional space with several peaks (multi-modal functions with peaks of equal width were searched in both of the above papers), the niche size is set to the width of the landscape divided by the number of peaks. This method of tuning the niche (1) violates the principle of isolating the landscape function in a "black box", and (2) is not applicable to more complex search spaces, where there is no definable best size. This paper will introduce a modification to sharing called *scheduled sharing* intended to remove the violation of the "black box" principle and make sharing applicable to more complex problems.

Fixed-niche sharing

One method of implementing fixed-niche sharing is described in [7, 5]. We largely followed their method in our implementation of sharing, which is described below.

In any genetic optimization problem, we evaluate at selected points in the search space some function embodying the parameters and constraints to be optimized. For instance, if our goal was to find the highest point on a rugged two-dimensional landscape, the *phenotype* of a single organism i would be p_i, an (x, y) pair; and the function $f(p_i) = f_i$ would yield the organism's phenotypic *fitness*, or the height of the landscape at that point.

In order to implement sharing, we will multiply f_i, the phenotypic fitness of organism i, by nsh_i, the *normalized sharing factor* calculated for the organism, to get g_i, the organism's overall fitness. The normalized sharing factor, nsh_i, is a number between zero and one and is generated in the following way.

We define a difference function for any two organisms i and j as

$$\begin{aligned} \mathrm{diff}(d_{ij}) &= \left[\frac{d_{ij}}{\sigma_{\mathrm{share}}}\right]^{\alpha} \text{ if } d_{ij} < \sigma_{\mathrm{share}} \\ &= 1 \text{ otherwise} \end{aligned}$$

where σ_{share} is the niche size and $d_{ij} = d(x_i, x_j)$, the distance between organisms i and j measured by some metric defined on the phenotype or genotype space. In our case, the phenotype was an (x, y) pair, so we used a two-dimensional cartesian distance function. (α was set to 1 in the results described below.) The function $\mathrm{diff}(d_{ij})$ is low when i and j are like each other, that is, when they share the same niche.

We next define a sharing factor sh_i for organism i relative to the rest of the population:

$$sh_i = \frac{\sum_j \mathrm{diff}(d_{ij})}{J}$$

sh_i is therefore low when organism i is competing intensely with its neighbors.

The *normalized sharing factor*, nsh_i, is found simply by normalizing and constraining the range of sh_i:

$$nsh_i = r \cdot \frac{sh_i}{\max_i sh_i} + (1 - r)$$

where r is a number $(0 \leq r \leq 1)$ that constrains the range of the normalized sharing factor, nsh_i. This limits the impact that the sharing factor can have on the phenotypic fitness. (In the results presented below, r was set to .75.)

The normalized sharing factor for organism i, nsh_i, will be a number $(1 - r) \leq nsh_i \leq 1$ that will be as high as 1 when organism i is not required to share at all, and will be as low as $(1 - r)$ when the organism finds itself sharing intensely with its many nearby neighbors in the phenotype space.

Finally, we multiply f_i by nsh_i to find g_i, the overall fitness function for organism i:

$$g_i = nsh_i \cdot f_i$$

Past work on sharing

In past work, it has been demonstrated that organisms can be more evenly distributed across one-dimensional landscapes of five peaks of equal width when they are forced to share with a fixed niche size *tuned* to the particular width of the peaks [7, 5]. It is admitted in [5], however, that the niche size, σ_{share}, "must be set carefully." Sharing at a fixed niche size appropriate to a five-peak landscape would only slow the optimization of a unimodal function by distributing the trials more broadly than is necessary. A function with ten times as many peaks would be better helped by sharing with one-tenth the niche size. Moreover, a single best niche size cannot be meaningfully defined for more complex search spaces. A complex space may contain peaks of sizes varying by many orders of magnitude, over which it may be desirable to evenly distribute organisms using sharing.

Most previous studies of sharing in genetic algorithms have been limited in that they (1) required hand tuning of the size of the niche within which organisms were to share with one another, (2) tested their algorithms by optimization of simple one- or two-dimensional functions, and (3) simulated small populations. These three limitations are interrelated: although such hand tuning is feasible on simple functions, it is not possible on complex functions or for practical applications of optimization. In addition, more complex test functions with many local optima tend to require larger populations for their adequate solution; most of these studies were conducted without parallel computers and so used smaller populations.

2. Current modifications to sharing

The research discussed in this paper seeks to modify fixed-niche sharing to be compatible with more difficult and more realistic test functions. We call the primary modification *scheduled sharing*, in which the niche size is initially large and then slowly reduced according to a schedule.

In addition, we describe *sampled sharing*, a method to remove the $O(n^2)$ communications between processors required by standard sharing. The possibility of using sampling to speed up sharing was suggested, though not implemented, in [7].

These improvements to standard sharing are tested using populations of up to 65,536 organisms. An adequate trial for such large populations requires a more difficult search space than has been used in past work. We describe a method to generate rugged two-dimensional landscapes, with features on many scales that are exploitable by genetic search. These landscapes have many more local optima of different sizes, from global structures to structures 1.7×10^{-7} the area of the landscape. They replace the simple, single-feature-size landscapes used in past sharing work.

Scheduled sharing: generalization of fixed-niche sharing

This work introduces a technique called *scheduled sharing*, which begins a run of n generations with a large niche size and then decreases the niche size as a function of the percentage of the total run that has been completed. Starting with a large niche size causes all organisms to repel each other, spreading the organisms evenly across the space, and assures that the initial stage of the search is a broad surveying of the landscape. As the niche size shrinks, the organisms are allowed to cluster at a controlled rate in areas of the space that are of greater interest. This clustering continues until, in the last stages of the run, organisms cluster densely in the areas of greatest interest. This assures that the smallest local optima will be searched fully before the run is ended. The controlled relaxation to convergence is reminiscent of the process of simulated annealing.

We wanted our schedule to cause the algorithm to obey the principles of *attention focusing* [11], namely that it should initially encourage the algorithm to choose the generally optimal large-scale areas of the space, and then allow the focusing of attention first upon mid-sized and then upon smaller details within the selected areas. Mauldin has also tried a similar method in [10] by requiring a linearly decreasing minimum hamming distance between the genotypes of his population over time. We followed the example of Kirkpatrick's work on simulated annealing [9], choosing to equate the size of the niche to an exponentially decaying function of the percentage of the run completed:

$$\sigma_{\text{share}} = (\text{width of landscape}) \cdot 2^{-\beta \cdot \frac{(\text{current generation})}{(\text{max generation})}}$$

where β is a parameter controlling how quickly the exponential function drops off. (β was set to 14 in the results described below.) The choice of such a schedule is discussed fully in our results and conclusion.

Sampled sharing: fast sharing for large populations

Because such large populations were used in this work (up to 65,536 organisms), and because communication between processors is often the most expensive part of a parallel computation, we did not sum over all j in the calculation of sh_i, but rather chose a random subset of the js each generation, and summed over those. We call this technique *sampled sharing*. We sampled a fixed number across all population sizes (rather than the alternative of sampling a percentage of each population size), since, in the limit of a large population, the variance of the error caused by sampling is dependent upon S, the number of points sampled, rather than on S/P, the percentage of the total population that is sampled.

Formally, if $m_k(x)$ is the kth moment of the distribution in the variable x:

$$m_k(x) = \sum_s \frac{x_s^k}{S}$$

then the variance of the error in the calculation of sh_i, the sharing factor for organism i as defined above on a population size P using a sample size S is

$$\text{var} \leq \left\lceil \frac{P+S}{P \cdot S} \right\rceil \cdot [\, -4m_1(x)m_3(x) + 3m_2^2(x) + m_4(x) \\ -4m_1(y)m_3(y) + 3m_2^2(y) + m_4(y) \,]$$

In the limit of a large population, $P \to \infty$, this becomes

$$\text{var} \leq \left\lceil \frac{1}{S} \right\rceil \cdot [\, -4m_1(x)m_3(x) + 3m_2^2(x) + m_4(x) \\ -4m_1(y)m_3(y) + 3m_2^2(y) + m_4(y) \,]$$

In other words, for large populations the variance due to sampling is dependent only on the sample size S and on the moments (which depend on the distribution), not on the population size P or on S/P [3].

Although it is straightforward to calculate the variance, it is not obvious how large the variance would have to be to have a deleterious effect on the performance of the algorithm. Therefore, we chose S empirically, finding that sampling 164 points had no deleterious effect on the algorithm. The derivation above assures us that using the same number of points should yield the same variance over different populations.

3. Implementation

Scheduled sharing as described in this paper was implemented on the Connection Machine supercomputer, a massively parallel single-instruction multiple-data (SIMD) machine. We mapped each organism to one virtual processor. On the Connection Machine, a variable number of virtual processors can be mapped to each physical processor actually present in the machine, which allows easy change of population size. We used machines of up to 32,768 physical processors to simulate up to 65,536 organisms. For communication between processors, the machine can be configured as an n-cube of arbitrary dimension. We configured the machine as a two-dimensional grid and allowed organisms to mate locally on the grid: each participating organism's partner was a random, nearby neighbor (within 5 random steps in the x or y directions). This was arranged so that each organism had exactly one partner. Each pair of organisms had a 50% chance of mating and a 50% chance of competing by simple replacement of the less fit organism by the more fit. A fixed 2% mutation rate was used.

The optimization problem that our algorithm was made to solve was the search for maxima on various rugged two-dimensional landscapes, as described below. The organisms' genetic material therefore consisted of an x and a y chromosome. During crossover, the x and y chromosomes were lined up parallel to one another and the cross was made at a randomly selected point, the same point for both chromosomes. Each chromosome had 15 bits of accuracy, yielding a total of $2^{30} = 1.07$ billion points in a given landscape.

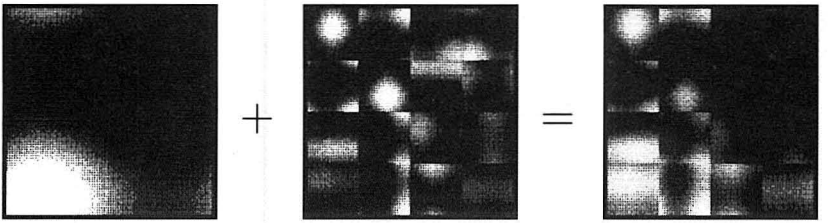

Figure 1: Subfunction S_0 plus subfunction S_1 yields landscape L.

Searching a rugged two-dimensional landscape

In past work concerned with the improvement of genetic algorithms there is significant precedent for using the optimization of one- and two-dimensional functions to study the performance of different algorithms, for example, [6, 7, 1]. Such functions have the advantage that they are easy to visualize, making it easier to speculate about why a particular change to an algorithm has improved or degraded performance.

Past work on sharing itself has used landscapes with five or fewer local optima, having equal-sized basins of attraction [7, 5]. This is understandable in that the intent of the research was to introduce sharing and show that it had a positive effect. Our intent was to investigate spaces where the basins of attraction of the local optima varied dramatically in size, and in which there were many more local optima. When searching such a space it is no longer possible simply to set the niche size to the size of the basin of attraction, since there is no single size. Problems containing significant numbers of local optima and discontinuities also more closely reflect practical applications of optimization.

Our two-dimensional test landscapes L were composed of the superposition of several two-dimensional trigonometric functions S_i of different amplitudes and frequencies. For instance, figure 1 shows two subfunctions S_0 and S_1, and the landscape L produced by their summation. In the figure, white areas represent relatively high points on the landscape, and black areas relatively low points. The actual landscapes we used, as described below, had up to five superimposed subfunctions with grid sizes many orders of magnitude smaller than those in the figure.

The subfunctions S_i had two basic components: a random-grid function, and the function $\cos(x) + \cos(y)$.

The random-grid function, $R_n(x, y)$, returns a randomly initialized, fixed number, $0 \leq a_{ij} \leq 1$, to all the (x, y) points within a given square (designated by the pair (i, j)) of an $n \times n$ grid superimposed on the domain.

We used R_n in combination with the $\cos(x) + \cos(y)$ function to construct subfunctions of different frequencies that were then summed to produce the final landscape. Within each subsquare of an $n \times n$ grid, the cosine functions were given a different random offset and amplitude in both the x and y directions with four copies of the R_n function, $R_n^1 \cdots R_n^4$ (the random

offset and amplitude are included to allow discontinuity in the space). The subfunctions S_i were defined as

$$S_i = R^1_{n_i} \cos(v_i x + R^2_{n_i}) + R^3_{n_i} \cos(v_i y + R^4_{n_i})$$

The final landscapes used to test the genetic algorithms were then

$$\begin{aligned} L &= \sum_i W_i S_i + L_0 \\ &= \sum_i W_i (R^1_{n_i} \cos(v_i x + R^2_{n_i}) + R^3_{n_i} \cos(v_i y + R^4_{n_i})) + L_0 \end{aligned}$$

where W_i are the weights dictating the contributions of the subfunctions at the different frequencies determined by the numbers v_i. L_0 is a constant set to $\sum_i 2W_i$, so that no point returns a negative fitness.

In the experiments described below we used an i of 5, building landscapes out of five subfunctions, each with structure of a different frequency. These ranged from global structures to tiny structures 1.7×10^{-7} the total area of the landscape. For a simple visualization, refer back to figure 1, depicting: subfunction S_0, which has global structure; subfunction S_1, which has structure on the scale of 1/16 the area of the landscape; and landscape L, the summation of the two subfunctions.

For a given set of weights W_i, an extremely large number of random landscapes can be generated. (We can for testing purposes, of course, generate the same one many times by starting with the same random seed.) In the set of all landscapes that could be randomly generated with a given set of weights W_i, not all landscapes will have the same global optimum; but the highest global optimum of any landscape in the set is equal to $2L_0$. Although the particular "easy" and "difficult" landscapes used in the experiments below may not have the same global optimum, and do not have the same sets of weights W_i, their respective sets of weights do sum to the same L_0.

The landscapes defined above satisfy our desire to control precisely the strengths and scales of the "hints" as to the location of probable optima that are available to the genetic algorithm. In addition, we can easily generate a large number of different landscapes that are like each other in the amount of exploitable information they provide. The landscapes are two-dimensional in this work, following precedent in genetic algorithms research for the use of one- and two-dimensional algorithms, and because two dimensions are so easily visualized; however, they could be expanded to higher dimensionality. One possible drawback of the landscapes is the expense of the cosine calculation, which may make the landscapes slower to generate than other test functions that could be defined. The cosine, however, is also easy to visualize and work with. One of the most important features of the landscapes is that any point can be calculated "on the fly" by any processor; that is, the points are not completely calculated once and stored, but rather each point is calculated locally as it is needed, saving both time and memory costs, which would be prohibitive for the vast landscapes of 1.07 billion points that we used.

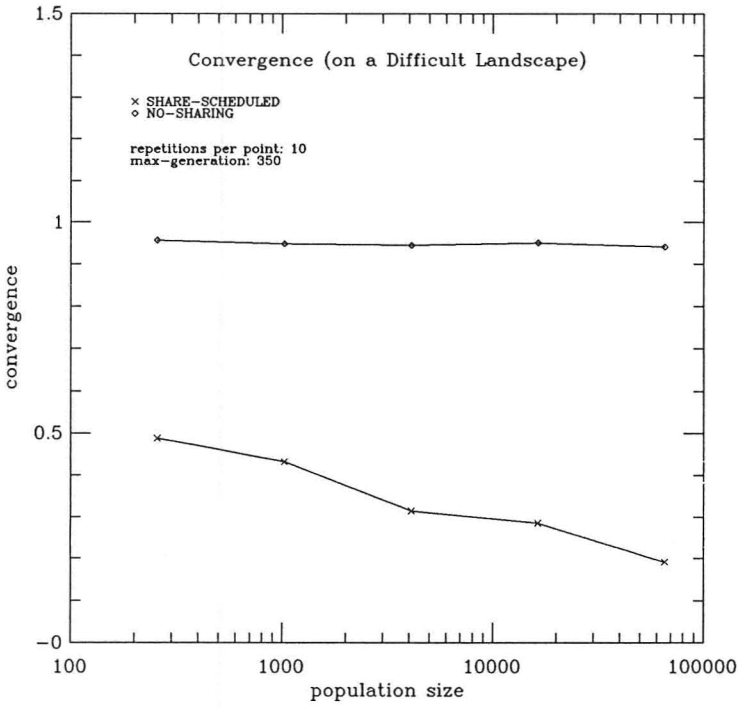

Figure 2: Convergence on a difficult landscape.

4. Results

Scheduled sharing inhibits convergence

The *convergence* of a population can be calculated by dividing the search space into many disjoint subspaces or "buckets," calculating x_i, the percentage of the total population within each bucket, and summing the squares of these percentages. This measure will be higher for a population that is more concentrated in one or a few buckets, and lower for populations more evenly distributed across the buckets. Therefore $\sum_i x_i^2$ is a useful measurement of the convergence of a population. (An alternative measure sometimes used is $\sum_i x_i \ln x_i$.) Convergence measurements given later in this document were performed with 256 buckets and give a rough measure of the self-similarity of the population.

As figure 2 shows, scheduled sharing effectively inhibits convergence. Although a non-sharing population will usually converge by generation n (where n is dependent on the problem, the implementation, and the population size), an algorithm using scheduled sharing can delay convergence for as long as is desired, and can therefore continue to search efficiently. Figure 2 shows the convergence after 350 generations for populations from 256 to 65,536

organisms (note that the population scale is logarithmic), with sharing and without sharing. Whereas populations of all sizes are near total convergence after 350 generations without sharing, all of the populations with sharing have much lower convergence.

Easy landscapes and difficult landscapes

In the following discussion we refer to two classes of landscapes, which can be constructed by setting the weights W_i of the subfunctions. *Easy* landscapes are those having a small number of local optima with wide basins of attraction. They can be constructed by giving a large weight to the contribution of low-frequency subfunctions and a lesser weight to the contribution of higher-frequency subfunctions. *Difficult* landscapes have a larger number of local optima with small basins of attraction. These landscapes are generated by weighting the contributions of higher-frequency subfunctions more than the contributions of lower-frequency subfunctions. Slowed convergence—and the broader surveying of the landscape that it implies—is most profitable on such difficult landscapes.

Performance improves for difficult landscapes and/or smaller populations

The scheduled delay of convergence effected by scheduled sharing, in certain cases, improves the overall performance of the search. (Overall performance is measured here by taking the average, over r runs, of the best organism found at any point during a run of n generations.) In particular, controlled convergence helped overall performance most when either the landscape was difficult (as described above), or the population size was small, or both.

To give some measure of the absolute optimization performance of the two genetic algorithms (scheduled sharing and non-sharing) on the landscapes, we include in figures 3 and 4 a comparison to a random search algorithm. At a given population, the random search algorithm performed the same number of fitness function evaluations as the two genetic algorithms to which it is compared.

Figure 3 shows overall performance of the scheduled sharing and non-sharing genetic algorithms, and of the random search algorithm, on a relatively difficult landscape, as defined above. The y axis shows overall performance, and the x axis shows the population size (again on a logarithmic scale). Note that scheduled sharing populations outperformed non-sharing populations for all population sizes, including the highest. We hypothesize that, for this more difficult landscape, the extra effort taken to broadly survey the entire landscape at the outset of the search, and then converge to interesting areas in a controlled fashion, allowed the system to achieve better overall performance.

The fact that the random search algorithm outperforms the non-sharing genetic algorithm for the two lowest populations gives an indication of just how difficult the landscape is: the non-sharing algorithm, at the two lowest

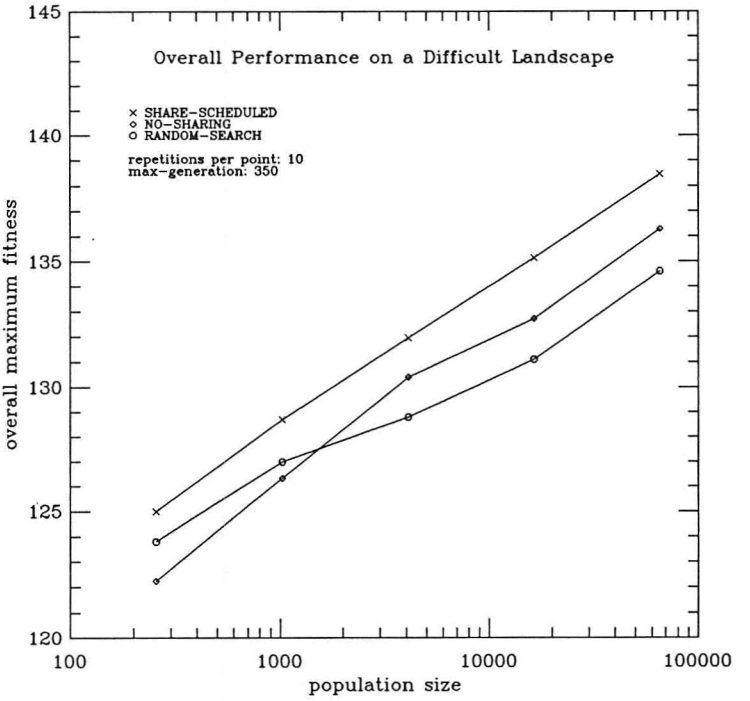

Figure 3: Performance on a difficult landscape.

populations, was unable to use the very few "hints" available in the structure of the landscape to beat a brute-force random search; it converged too quickly to lesser local optima. The scheduled sharing algorithm, however, whose delayed convergence encouraged a broader search of the difficult landscape, found enough hints to consistently outperform random search at all population sizes.

Note that, for the difficult landscape in figure 3, sharing populations of a given size do almost as well as non-sharing populations of four times the size. Sharing does require some extra computation—a generation with sharing in this case took about 1.5 to 2 times as long to compute as a non-sharing generation (though this will be problem- and implementation-dependent). This extra computational cost, however, is more than compensated for by the fact that only one-quarter the population needs to be used to get nearly the same performance.

We hypothesize that the control of convergence by scheduled sharing allows more efficient use of the available population. This ability to use a small population more efficiently is not such an advantage, however, if the population size is already large relative to the difficulty of the landscape. For instance, see figure 4, which shows overall performance versus

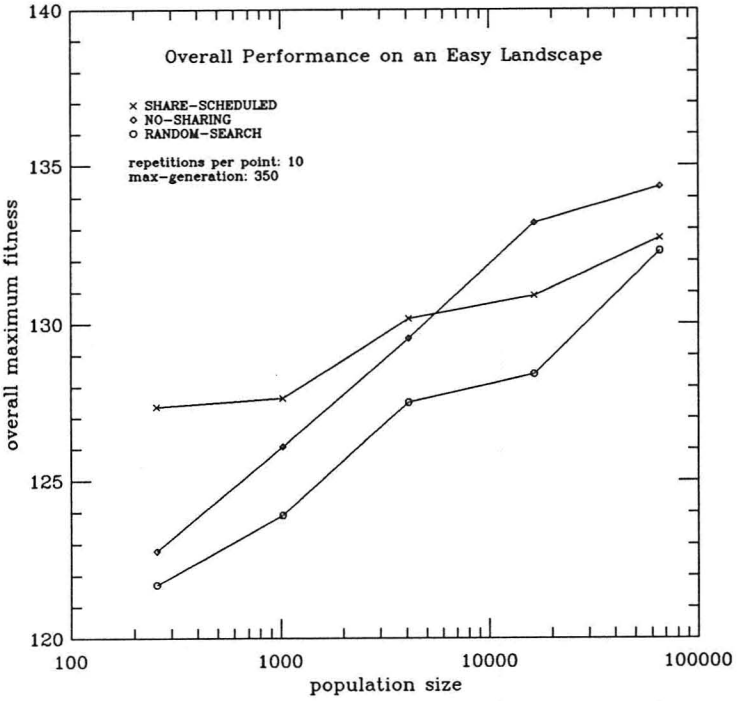

Figure 4: Performance on an easy landscape.

population size for an easier landscape. Note that, for all the lower populations, the scheduled sharing algorithm performs better than the non-sharing algorithm. However, at the two highest populations on this relatively easy landscape, the non-sharing algorithm outperforms the sharing algorithm. We hypothesize that the scheduled sharing algorithm becomes less efficient than the non-sharing algorithm by blindly forcing the population to adhere to its schedule, instead of letting it quickly optimize on the details of the landscape, which in this case would be sufficient. As a consequence, the scheduled sharing algorithm does not have as much time to fine-tune and search preferred areas intensively, and is outperformed in this case by the non-sharing algorithm. In addition, at the largest populations, the random initialization of the population probably provides to the non-sharing algorithm a fairly broad initial survey of the space (at least, sufficiently broad for the easy landscape), reducing the advantage gained by slowing convergence.

Both genetic algorithms beat the random search algorithm at all population sizes on the easy landscape because the easy landscape gives many exploitable hints to direct the genetic algorithms toward promising search areas. At the highest population, the random search algorithm approaches most nearly the performance of the scheduled sharing algorithm for two

reasons: (1) at the highest population, the scheduled sharing algorithm is hindered most by the improper schedule (since, for large populations, a forced broad survey is least needed); and (2) both of the genetic algorithms are leveling off near the global optimum of the easy landscape, so the random search algorithm is able to catch up.

5. Conclusions and future work

In this paper we hope to have given some direction to the study of sharing. We believe that the mechanism of a sharing schedule provides flexible control of the convergence of a genetic algorithm. It appears to be a good tool for the job; we need now to learn how better to use the tool.

In removing the need to tune the niche size to the problem at hand, we have created a need to tune the sharing schedule. This replacement is, however, an improvement: whereas it doesn't make sense to select a single fixed niche size for any but the simplest of problems (i.e., as those with "hints" only on one scale, such as a sine wave), it will probably be possible in future work to discover ways of tuning the schedule.

Our results have shown that scheduled sharing can help overall performance for difficult landscapes and/or populations that are small relative to the difficulty of the landscape. The results suggest that easier landscapes and/or larger populations would also run well if we could somehow detect when it is allowable to "turn off" the fixed schedule—to let the convergence of the algorithm speed up. Obviously, to make a scheduled sharing algorithm that is efficient at searching landscapes of widely varying difficulty we need a "difficulty detector" to decide upon the schedule.

Without violating the principle of the "black box" search function, it may be possible to introduce other information as feedback to define a problem-dependent sharing schedule. Instead of fixing the niche size to a function of "percentage of run completed," we could tie it dynamically to some other variable. For instance, the niche size could decrease as some function of the current population convergence. (Convergence has been used as feedback in [12].) Alternatively, the rate of improvement of the average performance might provide some information about the landscape: as improvement slows down, it implies that the population has efficiently searched the landscape at the current niche size, and the niche size should be decreased to allow exploration of detail. Other possibilities include the monitoring, and control through manipulation of the sharing schedule, of Baker's "percent involvement" predictor of rapid convergence [2]; or of an entropy measure like that of Wilson [13].

Any of the above feedback mechanisms could allow the niche size, instead of changing blindly as a function of time, to change dynamically in response to the behavior of the algorithm on a particular search space. These possibilities represent the most promising avenues for future work.

Acknowledgments

The calculation of variance for sampled sharing is due to Anand Bodapati, who gave freely of his expertise throughout this project. The authors would also like to thank Gary Drescher, Woody Lichtenstein, and David Waltz for their support and advice. The code for this work was built on top of the DARWIN package, a public-domain genetic algorithm "skeleton" for the Connection Machine System.

References

[1] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing* (Norwell, MA, Kluwer Academic Publishers, 1987).

[2] J. E. Baker, "Adaptive Selection Methods for Genetic Algorithms," pages 101–106 in *An International Conference on Genetic Algorithms*, edited by John J. Grefenstette (Hillsdale, NJ, Lawrence Erlbaum, 1988).

[3] A. Bodapati, personal communication (1991).

[4] L. Booker, "Improving Search in Genetic Algorithms," pages 61–73 in *Genetic Algorithms and Simulated Annealing*, edited by Lawrence Davis (Los Altos, CA, Morgan Kauffman, 1987).

[5] K. Deb and D. E. Goldberg, "An Investigation of Niche and Species Formation in Genetic Function Optimization," pages 42–50 in *Proceedings of the Third International Conference on Genetic Algorithms*, edited by J. David Schaffer (San Mateo, Morgan Kaufmann, 1989).

[6] K. A. DeJong, "An Analysis of the Behavior of a Class of Genetic Adaptive Organisms" (Doctoral dissertation, University of Michigan), *Dissertation Abstracts International*, *36*(10) (1975) 5140B (University Microfilms No. 76-9381).

[7] D. E. Goldberg and J. Richardson, "Genetic Algorithms with Sharing for Multimodal Function Optimization," pages 41–49 in *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* edited by John Grefenstette (Hillsdale, NJ, Lawrence Erlbaum Associates, 1987).

[8] D. E. Goldberg, "Sizing Populations for Serial and Parallel Genetic Algorithms," pages 70–79 in *Proceedings of the Third International Conference on Genetic Algorithms*, edited by J. David Schaffer (San Mateo, Morgan Kaufmann, 1989).

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, **220** (1983) 671–680.

[10] M. L. Mauldin, "Maintaining Diversity in Genetic Search," Internal working document, Carnegie Mellon Department of Computer Science (1984).

[11] M. E. Palmer, "The Airplane-finder Project, or, An 'Attention Focusing' Network for Pattern Recognition on 'Clusterable' Data Spaces," pages 45–48 in *Proceedings of the 1990 Long Island I.E.E.E. Student Conference on Neural Networks*, edited by Frank P. Li (NYIT, 1990).

[12] C. G. Shaefer, "The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique," pages 50–55 in *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, edited by John Grefenstette (Hillsdale, NJ, Lawrence Erlbaum Associates, 1987).

[13] S. W. Wilson, "Classifier Systems and the Animat Problem," pages 199–228 in *Machine Learning*, Volume 2 (San Mateo, Morgan Kauffman, 1986).