A Survey of Parallel Genetic Algorithms

Erick Cantú-Paz

IlliGAL Report No. 97003
May 1997

Revised version of IlliGAL Report No. 95007.

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-0897
Fax: (217) 244-5705

A Survey of Parallel Genetic Algorithms

Erick Cantú-Paz
Department of Computer Science and
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
cantupaz@illigal.ge.uiuc.edu

Abstract

Genetic algorithms (GAs) are powerful search techniques that are used successfully to solve problems in many different disciplines. Parallel GAs are particularly easy to implement and promise substantial gains in performance and as such there has been extensive research in this field. This paper attempts to collect, organize, and present in a unified way some of the most representative publications on parallel genetic algorithms. To organize the literature, the paper presents a categorization of the techniques used to parallelize GAs and shows examples of all of them. However, since the majority of the research in this field has concentrated on coarse-grained parallel GAs this survey focuses on this type of algorithms. Also, the paper describes some of the most significant problems in modeling and designing coarse-grained parallel GAs and presents some recent advances.

1 Introduction

Genetic Algorithms (GAs) are efficient search methods based on principles of natural selection and genetics. They are being applied successfully to find acceptable solutions to problems in business, engineering, and science (Goldberg, 1994). GAs are generally able to find good solutions in reasonable amounts of time, but as they are applied to larger and harder problems there is an increase in the time required to find adequate solutions. As a consequence of this there have been multiple efforts to make GAs faster, and one of the most promising choices is to use parallel implementations.

The objective of this paper is to organize and present the most relevant publications on parallel GAs. In order to organize the growing literature in this field we present a categorization of the different types of parallel implementations of GAs. By far, the most popular parallel GAs consist on multiple populations that evolve separately most of the time and exchange individuals occasionally. This type of parallel GAs is called coarse-grained or distributed GAs and this survey concentrates on this class of implementation. However, this paper also describes the other major types of parallel GAs and discusses briefly some examples.

There are many examples in the literature where parallel GAs are applied to find solutions for a particular problem, but the intention of this paper is not to enumerate all the problems where parallel GAs have been successful in finding good solutions, but instead to highlight those publications that have contributed to the growth of the field of parallel GAs in some ways. The survey examines in more detail those publications that introduce something new or that attempt to explain why this or that works, but we also mention a few of the application problems to show that parallel GAs are useful in the "real world" and are not just a curiosity.

This paper is organized as follows. The next section is a brief introduction to genetic algorithms and section 3 describes our categorization of parallel GAs. The review of parallel GAs begins in section 4 with a presentation of early publications. We describe the global method to parallelize GAs in section 5. Sections 6 and 7 survey the literature on coarse-grained parallel GAs and section 8 summarizes the research on fine-grained parallel GAs. Section 9 deals with hybrid methods of parallelizing GAs. Next, section 10 describes some open problems in modeling and design and some recent advances. We end this report with a summary and the conclusions of this project.

2 Genetic algorithms — A quick introduction

This section intends to provide basic background material on GAs. It defines some terms and describes how a simple GA works, but it is not a complete tutorial. Interested readers can consult the book by Goldberg (1989) for very detailed background information on GAs.

Simple genetic algorithms are based on a basic evolutionary principle: better individuals survive and reproduce more often than other individuals. To implement this principle a GA maintains a population of candidate solutions that evolves over time and ultimately converges to a unique solution. The individuals in the population of the GA are represented with a **chromosome** that encodes the variables of the problem that is attempted to be solved. Most often there is a single chromosome and it consists on fixed-length binary string, but there are GAs with diploid (two chromosomes) individuals and that use strings from higher-cardinality alphabets and with variable length. Each individual has a numeric **fitness** value that measures how well the parameters encoded in it solve the problem. Frequently, GAs are used as optimizers and the fitness of an individual is determined as the value of the objective function at the point represented by the individual.

Genetic algorithms have a **selection** mechanism that identifies the fittest individuals of the current population to serve as parents of the next generation. The selection algorithm can take many forms, but it always ensures that the best individuals have a higher probability to reproduce and breed to form a new generation.

To explore the search space simple GAs use two operators based loosely on natural genetics: crossover and mutation. The primary exploration operator for GAs is **crossover** which consists on choosing randomly a pair of individuals among those selected previously and swap substrings between them with a certain probability. This exchange of substrings occurs around crossing points that are selected randomly.

The **mutation** operator is usually considered a secondary search operator. The main function of mutation is to restore diversity that may be lost from the repeated application of selection and crossover. This operator takes one string from the population and alters randomly some value within it. The probability of applying mutation is very low, just as it is in Nature. In contrast, the probability of crossover is usually high.

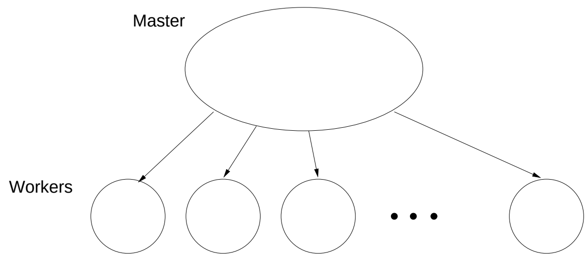

Figure 1: A schematic of a global parallel GA.

It has been determined that the time that a GA takes to converge to a unique solution depends on the size of the population. Goldberg and Deb (1991) computed the time it takes for one solution to take over the entire population using different selection methods. They determined that the faster selection method results in a convergence time of $O(n \log n)$, where n is the size of the population.

Genetic algorithms are not *guaranteed* to find an *optimal* solution and that their effectiveness is determined largely by the size of their population (Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997). As the population size increases the GA has a better chance of finding the global solution, but as we saw above the computation cost also increases as a function of the population size. There is a delicate balance between the quality of the solution and the time that a simple GA needs to find it. This balance also exists for parallel GAs and we shall see later how it affects some design choices, but lets first examine in the next section a way to organize the extensive literature on parallel GAs.

3 A classification of parallel GAs

The basic idea behind many parallel programs is to divide a task into chunks and solve the chunks simultaneously using multiple processors. This divide-and-conquer approach can be used in different ways and this leads to different methods to parallelize GAs. Some methods change the behavior of the algorithm while others do not. Some methods can exploit massively parallel computer architectures while others are better suited to multicomputers with fewer and more powerful processing elements. The classification of parallel GAs that is presented in this section is similar to other classifications (Adamidis, 1994; Gordon & Whitley, 1993; Lin, Punch, & Goodman, 1994), but we extend them to include one more category.

The first method to parallelize GAs is to do a **global** parallelization. In this type of parallel GAs there is only one population as in the serial GA, but the evaluation of individuals and the genetic operators are parallelized explicitly (see figure 1). Since there is only one population, selection considers all the individuals and every individual has a chance to mate with any other (i.e., there is **random mating**), and therefore the behavior of the algorithm remains unchanged. This method is relatively easy to implement and a significant speedup can be expected if the communications cost does not dominate the computation cost (Cantú-Paz, 1997). In section 5 we describe strategies for implementing this model in shared and distributed memory machines.

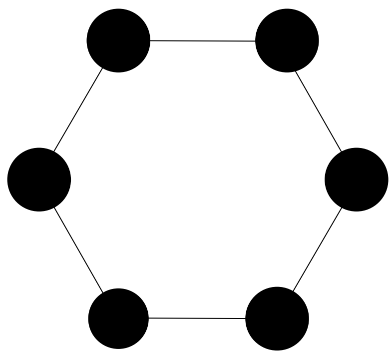

Figure 2: A schematic of a coarse-grained parallel GA.

A more sophisticated idea is used in **coarse-grained** parallel GAs. In this case the population of the GA is divided into multiple subpopulations or **demes** that evolve isolated from each other most of the time, but exchange individuals occasionally. This exchange of individuals is called **migration** and, as we shall see in later sections, it is controlled by several parameters. Coarse-grained parallel GAs introduce fundamental changes in the operation of the GA and have a different behavior than simple GAs (see figure 2).

Sometimes coarse-grained parallel GAs are known as "distributed" GAs because they are usually implemented on distributed-memory MIMD computers. Coarse-grained GAs are also known as "island" parallel GAs because in Population Genetics there is a model for the structure of a population that considers relatively isolated demes, and it is called the *island model*.

It seems that since the size of the demes is smaller than the population used by a serial GA, we would expect that the parallel GA will converge faster. However, when we compare the performance of the serial and the parallel algorithms we must also consider the quality of the solutions found in each case. It is true that a smaller deme will converge faster, but it is also true that the quality of the solution might be poorer. Section 11 presents a recent theory that predicts the quality of the solutions for some extreme cases of coarse-grained parallel GAs and let us do fair comparisons with serial GAs.

The third approach in parallelizing GAs uses fine-grained parallelism. **Fine grained** parallel GAs partition the population into a large number of very small subpopulations (see figure 3). Indeed, the ideal case is to have just one individual for every processing element available. This model is suited for massively parallel computers, but it can be implemented on any multiprocessor.

It is important to emphasize that while the global parallelization method does not affect the behavior of the algorithm, the last two methods introduce fundamental changes in the way the GA works. For example, in the global method the selection operator takes into account the entire population, but in the other parallel GAs selection is local to each deme. Also, in the methods that divide the population it is only possible to mate with a subset of individuals (the deme), whereas in the global model it is possible to mate with any other individual.

The final method to parallelize GAs uses some combination of the first three. We

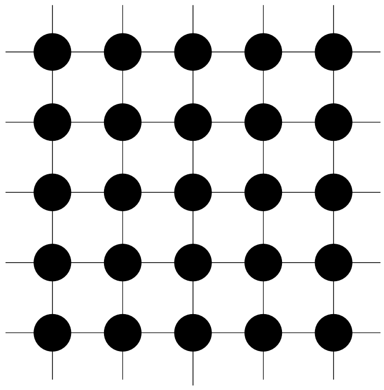

Figure 3: A schematic of a fine-grained parallel GA.

call this class of algorithms **hybrid** parallel GAs. Combining parallelization techniques results in algorithms that combine the benefits of their components and promise better performance than any of the components alone. In section 10 we review the results of some implementations of hybrid parallel GAs.

4 Early studies

Despite all the growing attention that parallel computation has received in the last 15 years, the idea of building massively parallel computers is much older. For example, in the late 1950s John Holland proposed an architecture for parallel computers that could run an undetermined number of programs concurrently (Holland, 1959; Holland, 1960). The hardware of Holland's computer was composed of relatively simple interconnected modules and the programs running on these machines could modify themselves as they were executing, mainly to adapt to failures in the hardware. One of the applications that Holland had in mind for this class of computers was the simulation of the evolution of natural populations.

The "iterative circuit computers", as they were called, were never built, but the hardware of some massively parallel computers of the 1980s (e.g., Connection Machine 1) use the same modular concept of Holland's computers. The software, however, is not very flexible in the newer computers.

An early study of how to parallelize genetic algorithms was conducted by Bethke (1976). He described (global) parallel implementations of a conventional GA and of a GA with a generation gap (i.e., it only replaces a portion of its population in every generation) and showed a very detailed analysis of the efficiency of the use of the processing capacity. His analysis showed that parallel GAs have an efficiency close to 100% in SIMD computers, but Bethke ignores the communications overhead. He also analyzed gradient-based optimizers and identified some bottlenecks that limit their parallel efficiency.

Another early study on parallel GAs was made by Grefenstette (1981). He proposed four prototypes for parallel GAs. The first prototype is a global parallel GA with a "master" processor that does selection and applies crossover and mutation. The individuals are sent to "slave" processors to be evaluated and return to the master at the end of every generation. The second prototype is very similar to the first, but there is no clear division between generations, when any slave processor finishes evaluating an individual it returns it to the master and receives another. This scheme eliminates the need to synchronize every generation and can maintain a high level of processor utilization, even if the slave processors operate at different speeds. The third prototype is also a global GA in which the population is kept in shared memory which can be accessed independently by slaves. The difference with the first two prototypes is that this one assumes the existence of a global shared memory.

Grefenstette's fourth prototype is a conventional coarse-grained parallel GA where the best individuals are broadcast every generation to all the other processors. The complexity of coarse-grained parallel GAs was evident from this early proposal and Grefenstette raised several "interesting questions" about the frequency of migration, the destination of the migrants (topology), and the effect of migration on preventing premature convergence. This is the only prototype that was implemented some time later (Pettey, Leuze, & Grefenstette, 1987b; Pettey, Leuze, & Grefenstette, 1987a) and we review some results in section 6.

Grefenstette also hinted at the possibility of combining the fourth prototype with any of the other three, creating a hybrid parallel GA.

5 Global parallelization

In this section we review in detail the global parallelization method. This method maintains a single population and the evaluation of the individuals and/or the application of genetic operators are done in parallel. Just like in the serial GA, each individual competes with all the other members of the population and also has a chance to mate with any other individual. In other words, selection and mating are still *global*, hence the name for the model.

The most common operation that is parallelized is the evaluation of the individuals, because the fitness of an individual is independent from the rest of the population and there is no need to communicate during this phase. The evaluation of individuals is parallelized by assigning a fraction of the population to each of the processors available. Communication occurs only as each processor receives its subset of individuals to evaluate and when the processors return the fitness values.

If the program stops and waits to receive the fitness values for all the population before proceeding into the next generation, then the algorithm is **synchronous**. A synchronous global GA has exactly the same properties as a simple GA, with the possibility of better performance being the only difference. However, it is also possible to implement **asynchronous** global GAs where the master does not stop to wait for any slow processors, but asynchronous global GAs do not work exactly like a simple GA. Most global parallel GA implementations are synchronous and in the rest of the paper we assume that global parallel GAs do the exact same search of simple GAs.

The global parallelization model does not assume anything about the underlying computer architecture, and it can be implemented efficiently on shared-memory and distributed-memory computers. On a shared memory multiprocessor, the population could be stored in shared memory and each processor can read the individuals assigned to it and write the evaluation results back without any conflicts[1].

The number of individuals assigned to any processor can be constant, but in some cases (like in a multiuser environment where the utilization of processors is variable) it may be necessary to balance the computational load among the processors using a dynamic scheduling algorithm (e.g., guided self-scheduling).

On a distributed memory computer, the population can be stored in one processor. This "master" processor would be responsible for sending the individuals to the other processors (the "slaves") for evaluation, collecting the results, and applying the genetic operators to produce the next generation.

In any case, the cost of communications is similar in shared and distributed-memory computers, the difference is that on the latter communications have to be specified explicitly. As more slaves are used, the cost of communications increases, but also the computations that each slave has to make decreases. This tradeoff has been analyzed recently (Cantú-Paz, 1997) and the results indicate that there is an optimal number of slaves that minimizes the execution time of global parallel GAs.

An example of global parallelization is the work of Fogarty and Huang (1991). Their goal was to evolve a set of rules for a pole balancing application which takes a considerable time to simulate. They used a network of transputers which are microprocessors designed specifically for parallel computations. A transputer can connect directly to only four transputers, and when there is a need to communicate with another node in the network, all the intermediate transputers must receive and retransmit the message. This causes an overhead in communications and to try to minimize it Fogarty and Huang connected the transputers in different topologies, but they concluded that the configuration of the network did not make a significant difference. They obtained reasonable speedups and identified the fast-growing communications overhead as an impediment for further improvements in speed.

Abramson and Abela (1992) implemented a global GA on a shared memory computer (an Encore Multimax with 16 processors) to search for efficient timetables for schools. They reported limited speedups, and blamed a few sections of serial code on the critical path of the program for the results. Later, Abramson, Mills, and Perkins (1993) added a distributed memory machine (a Fujitsu AP1000 with 128 processors) to the experiments, changed the application to train timetables, and modified the code. This time, they reported significant (and almost identical) speedups for up to 16 processors on the two computers, but the speedups degraded significantly as more processors are added, mainly due to the increase in communications.

A more recent implementation of a global GA is the work by Hauser and Männer (1994). They used three different parallel computers, but only got good speedups on a NERV multiprocessor (speedup of 5 using 6 processors), that has a very low communications overhead. They explained that they did not get good speedups on the other systems they used (a SparcServer and a KSR1) because they did not have complete control over the scheduling of computation threads to processors and their systems sometimes made inadequate decisions.

[1]We mean that there are no conflicts with other processors to access the same memory locations and thus no synchronization is required in this step, but there may be conflicts in the interconnection network that may slow the algorithm.

Besides the evaluation of individuals, other aspect of GAs that can be parallelized is the application of the genetic operators. Crossover and mutation could be parallelized using the same idea of partitioning the population and distributing the work among multiple processors. However, these operators are so simple that it is very likely that the time required to send individuals back and forth would offset any performance gains.

The communication overhead is also a problem when selection is parallelized because several forms of selection need information about the *entire* population and thus require some communications. Recently, Branke, Andersen, and Schmeck (1997) parallelized different types of global selection on a 2-D grid of processors and showed that their algorithms were optimal for the topology of the interconnection network of the computer that they used (their algorithms require $O(\sqrt{n})$ time steps on a $\sqrt{n} \times \sqrt{n}$ grid).

In conclusion, global parallel GAs are easy to implement and they can be a very efficient method of parallelization when the evaluation needs considerable computations. Besides, the method has the advantage of preserving the search behavior of the GA, so we can apply directly all the theory for simple GAs.

6 Coarse grained parallel GAs — The first generation

The important characteristics of coarse-grained parallel GAs are the use of few relatively large demes and migration. Coarse grained GAs are the most popular parallel method and many papers have been written describing innumerable aspects and details of their implementation. Obviously, it is impossible to review all these papers, and therefore this section presents only the most influential papers and a few relevant examples of particular implementations and applications.

Probably the first systematic study of coarse-grained parallel GAs was Grosso's dissertation (Grosso, 1985). His objective was to simulate the interaction of several parallel subcomponents of a population in evolution. Grosso simulated diploid individuals (so there were two subcomponents for each "gene") and the population was divided into five demes. Each deme exchanged individuals with all the other demes with a fixed migration rate.

With controlled experiments, Grosso found that the improvement of the average population fitness was faster in the smaller demes than in a single large panmictic population. This confirmed a long-held principle in population genetics that favorable changes in the genetic composition of an individual spread faster to the other individuals when the demes are small than when the demes are large. However, Grosso observed that when the demes were isolated, this rapid rise in fitness stopped at a lower fitness value than with the larger population. In other words, the quality of the solution found after convergence was worse in the isolated case than in the single population. With a low migration rate, the demes still behaved independently and explored different regions of the search space. The migrants did not have a significant effect on the receiving deme and the quality of the solutions was similar to the case where the demes were isolated. But, at intermediate migration rates the divided population found solutions similar to those found in the panmictic population. These observations indicate that there is a critical migration rate below which the performance of the algorithm is obstructed by the isolation of the demes, and above which the partitioned population may find solutions of the same quality as the panmictic population.

In a similar implementation of a parallel GA by Pettey, Leuze, and Grefenstette (1987a) a copy of the best individual found in each deme was sent to all its neighbors after every generation. The purpose of this constant communication was to ensure a good mixing of individuals. Like Grosso, the authors of this paper observed that parallel GAs with such a high level of communication are equivalent to a sequential GA with a single large panmictic population. This observations prompted other questions that remain unanswered today: (1) what is the level of communication necessary to make a parallel GA behave like a panmictic GA? (2) what is the cost of this communication? and (3) is the communication cost small enough to make this a viable alternative for the design of parallel GAs? More research is necessary to understand the effect of migration on the quality of the search in parallel GAs in order to be able to answer completely these questions.

It is interesting to realize that such important observations were made so long ago, at the same time that other systematic studies of parallel GAs were underway. For example, Tanese (1987) proposed a parallel GA with the demes connected on a 4-D hypercube topology. In Tanese's algorithm migrations occurred at fixed intervals between neighbor processors along one dimension of the hypercube. The migrants were chosen probabilistically from the best individuals in the subpopulation and they replaced the worst individuals in the receiving deme. In the first set of experiments Tanese fixed the interval between migrations to 5 generations and varied the number of processors and the migration rate (the percentage of the population that migrated). The parallel GA found results of the same quality as the serial GA with the two migration rates that she used and with the different processor counts. However, it is difficult to see from the experimental results if the parallel GA found the solutions sooner than the serial GA because the range of the times is too large. In the second set of experiments Tanese used different mutation and crossover rates in each deme attempting to overcome the problem of finding good values for the parameters of the GA to balance exploration and exploitation. The third set of experiments studied the effect of the exchange frequency on the search and the results showed that migrating too frequently or too infrequently degraded the performance of the algorithm.

Also in this year, Cohoon, Hegde, Martin, and Richards (1987) implemented a parallel GA and noted that there were some similarities with the theory of *punctuated equilibria*. This theory was proposed by Elredge and Gould to explain the (apparently) missing links in the fossil record. It states that most of the time populations are in equilibrium (i.e., there are no significant changes in its genetic composition), but some changes on the environment can start a rapid evolutionary change. One important part of the environment of a population is its own composition, because individuals in a population must compete for resources with the other members, and thus, the arrival of different individuals from other populations can punctuate the equilibrium and trigger evolutionary changes. Cohoon et al. noticed in their experiments with the parallel GA that in the time between migrations there was relatively little change in the demes, but new solutions were found shortly after migration.

Cohoon et al. used a linear placement problem as a benchmark and experimented connecting the demes using a mesh topology. However, they mentioned that the choice of topology is probably not very important in the performance of the parallel GA as long as it has "high connectivity and a small diameter to ensure adequate 'mixing' as time progresses". Note that the mesh topology they chose is densely connected and also has a small diameter ($O(\sqrt{r})$, where r is the number of demes).

This work was later extended by the same group using a VLSI application (a graph partitioning problem) on a 4-D hypercube topology (Cohoon, Martin, & Richards, 1991a;

Cohoon, Martin, & Richards, 1991b). Like in the mesh topology, the nodes in the hypercube have four neighbors, but the diameter of the hypercube is shorter than the mesh ($\log_2 r$).

Tanese (1989a) continued her work making a very complete experimental study on the frequency of migrations and the number of individuals exchanged each time. She compared the performance of a serial GA and parallel GAs with and without communication. All the algorithms had the same total population size (256 individuals) and executed for 500 generations. Tanese found that a coarse-grained GA with no communication and r demes with n individuals each could find —at some point during the run— solutions of the same quality as a single serial GA with a population of rn individuals. However, the average quality of the *final* population was much lower in the parallel isolated GA. When migration was used the final average quality increased and Tanese claimed that in some cases it was better than the final quality in a serial GA. However, this result should be interpreted carefully because the serial GA did not converge fully at the end of the 500 generations and some further improvement was still possible. More details about the experiments and the conclusions derived from this study can be found in Tanese's dissertation (Tanese, 1989b).

A recent paper by Belding (1995) confirmed Tanese's work using the Royal Road functions. Migrants were sent to a random destination, rather than using a hypercube topology, as in Tanese's original study, but the author claimed that experiments with a hypercube yielded similar results. In most cases, the global optimum was found more often when migration was used than in the completely isolated cases.

Even at this point, where we have reviewed only a handful of examples of coarse grained-parallel GAs, we may hypothesize several reasons that make these algorithms so popular:

- Coarse-grained GAs seem like a simple extension of the serial GA. The recipe is simple: take a few conventional (serial) GAs, run each of them on a node of a parallel computer, and at some predetermined times exchange a few individuals.
- Coarse-grain parallel computers are easily available, and even if there is no parallel computer available it is easy to simulate one with a network of workstations or even in a single-processor machine using free software (like MPI or PVM).
- There is relatively little extra effort needed to convert a serial GA into a coarse-grained parallel GA. Most of the program of the serial GA remains the same and only a few subroutines need to be added to implement migration.

In this section we reviewed some of the papers that initiated the systematic research on coarse-grained parallel GAs and that raised many of the important questions that this field still faces today. In the next section we review more papers about coarse-grained parallel GAs that show how the field matured and began to explore other questions about these algorithms.

7 Coarse-grained parallel GAs — The second generation

From the papers examined in the previous section, we can recognize that some very important issues are emerging. First, parallel GAs are very promising in terms of the gains in performance. Second, parallel GAs are more complex than their serial counterparts. For example, the migration of individuals from one deme to another is controlled by several parameters like: (a) the **topology** that defines the connections between the subpopulations, (b) a **migration rate** that controls how many individuals migrate, and (c) a **migration interval** that affects how often migrations occur.

In the late 1980s and early 1990s the research on parallel GAs began to explore alternatives to make parallel GAs faster and to understand better how they worked. Around this time the first theoretical studies on parallel GAs began to appear and the empirical research attempted to identify favorable parameters. In this section we review some of that early theoretical work and some studies on migration and topologies. Also, more researchers started to use coarse-grained parallel GAs to solve application problem and this section ends with a brief review of some of their work.

One of the directions in which the field matured is that parallel GAs began to be tested with very large and difficult test functions. A remarkable example of this is the work by Mühlenbein, Schomisch, and Born (1991) who described a parallel GA that found the global optimum of several functions that are used as a benchmark for different optimization algorithms. Later on, the functions used in this paper were adopted by other researchers as benchmarks for their own empirical work (e.g. (Chen, Meyer, & Yackel, 1993; Gordon & Whitley, 1993)). Mühlenbein, Schomisch, and Born included a local optimizer in their algorithm, and although they designed an efficient and powerful algorithm, it is not clear if the results obtained can be credited to the distributed population or to the local optimizer.

Mühlenbein believes that parallel GAs can be useful in the study of evolution of natural populations (Mühlenbein, 1989a; Mühlenbein, 1989b; Mühlenbein, 1992; Mühlenbein, 1991). This view is shared by others that perceive that a partitioned population with an occasional exchange of individuals is a closer analogy to the natural process of evolution than a single large population. But, while it is true that in a parallel GA we can observe some phenomena that have a direct counterpart in Nature (for example, we can observe how a favorable change in a chromosome spreads faster in smaller demes than in large demes) the main intention of the majority of the researchers in this field is to design faster search algorithms.

7.1 Early theoretical work

All the publications reviewed so far base their conclusions on the results of experiments. However, a theoretical understanding of (parallel) GAs is necessary to achieve a deeper understanding of these algorithms so we can utilize them at their full potential. How can we design better algorithms if we do not know what are the factors that limit their efficiency and accuracy?

Probably the first attempt to provide some theoretical foundations to the performance of a parallel GA was a paper by Pettey and Leuze (1989). In this article they described a derivation of the schema theorem for parallel GAs where randomly selected individuals are broadcast every generation. Their calculations showed that the expected number of trials allocated to schemata can be bounded by an exponential function, just as for serial GAs.

A very important theoretical question is to determine if (and when) a parallel GA can find a better solution than a serial GA. Starkweather, Whitley, and Mathias (1991) made two important observations regarding this question. First, it can be expected that relatively isolated demes will converge to different solutions and that migration and recombination will combine the partial solutions. Starkweather et al. speculate that because of this a parallel GA with low migration may find better solutions for separable functions. Their second observation is that if the recombination of partial solutions results in individuals with lower fitness (i.e., the function is not separable) then the serial GA might have an advantage.

7.2 Migration

Another sign of maturity of the field is that the research started to focus more in only one aspect of parallel GAs. In this section we review some publications that explore alternative migration schemes to try to make parallel GAs more efficient.

In the majority of coarse-grained parallel GAs, migration is **synchronous** meaning that it occurs at predetermined constant intervals, but it can also be **asynchronous** so that there is communication between demes only after some events occur. An example of asynchronous migration can be found in Grosso's dissertation where he experimented with a "delayed" migration scheme, where migration was enabled until the population was close to converge.

Braun (1990) used the same idea and presents an algorithm where migration occurs after the demes converge completely (the author uses the term "degenerate") with the purpose of introducing diversity into the demes to try to alleviate the problem of converging prematurely to a solution of low quality. The same strategy for migration was used later by Munetomo, Takai, and Sato (1993) and recently Cantú-Paz and Goldberg (1997a) presented theoretical models that predict the quality of the solutions found with this method of migration using a fully connected topology.

There is another interesting question being raised here: when is the right time to migrate? If migration occurs too early during the run, the number of correct building blocks in the migrants may be too low to influence the search on the right direction and we would be wasting expensive communication resources.

A different approach to migration was developed by Marin, Trelles-Salazar, and Sandoval (1994). They proposed a centralized scheme where the slaves execute a GA on their population and periodically send their best partial results to a master process. Then the master process chooses the fittest individuals found so far and broadcasts them to the slave nodes. Experimental results on a network of workstations showed that near-linear speedups can be obtained with a small number of nodes (around six), and the authors claim that their approach can scale to larger networks because communications are infrequent.

7.3 Communication topologies

A traditionally neglected component of parallel GAs is the topology of the interconnection between demes. The topology is an important factor in the performance of the parallel GA because it determines how fast (or how slow) a good solution disseminates to other demes. If the topology has a dense connectivity (or a short diameter, or both) good solutions will spread fast to all the demes and may quickly take over the population. On the other hand, if the topology is sparsely connected (or has a long diameter), solutions will spread slower and the demes will be more isolated from each other, permitting the appearance of different solutions. These solutions may come together at a later time and recombine to form potentially better individuals.

The communication topology is also important because it is a major factor in the cost of migrations. For instance, a densely connected topology may promote a better mixing of individuals, but it also entails higher communication costs.

The general trend on coarse-grained parallel GAs is to use **static** topologies that are specified at the beginning of the run and remain unchanged. Many implementations of parallel GAs with static topologies use the topology of the communications network of the computer available to the researchers. For example, implementations on hypercubes (Cohoon, Martin, & Richards, 1991a; Tanese, 1989a; Tanese, 1989b) and rings (Gordon & Whitley, 1993) are common.

A more recent empirical study showed that parallel GAs with dense topologies find the global solution using fewer function evaluations than GAs with sparsely connected ones (Cantú-Paz & Mejía-Olvera, 1994). This study considered fully connected topologies, 4-D hypercubes, a 4×4 toroidal mesh and unidirectional and bidirectional rings.

The other major choice on communication topologies is to use a **dynamic** topology. In this type of topology a deme is not restricted to communicate only with some fixed set of demes, but instead it sends its migrants to those demes that meet some criteria. The motivation behind dynamic topologies is to identify those demes where the migrants are likely to produce some effect. Typically, the criteria used to choose a deme as a destination includes measures of the diversity of the population (Munetomo, Takai, & Sato, 1993) or a measure of the genotypic distance between the two populations (or some representative individual of a population, like the best) (Lin, Punch, & Goodman, 1994).

7.4 Applications

Many application projects using coarse-grained parallel GAs have been published. In this section we mention only a few representative examples.

The graph partitioning problem has been a popular application of coarse- grained GAs (see for example the work by Béssiere and Talbi (1991), Cohoon, Martin, and Richards (1991c)) and Talbi and Bessière (1991)). The work of Levine (1994) on the this problem is outstanding. His algorithm found global solutions of problems with 36,699 and 43,749 integer variables. He found that performance, measured as the quality of the solution and the iteration on which it was found, increased as more demes were added. Also, Li and Mashford (1990) and Mühlenbein (1989b) found solutions for the quadratic assignment problem, another combinatorial optimization application.

Coarse-grained parallel GAs have also been used to find solutions for the problem of distributing computing loads to the processing nodes of MIMD computers by Neuhaus (1991), Seredynski (1994), and by Muntean and Talbi (1991).

Another challenging application is the synthesis of VLSI circuits and Davis, Liu, and Elias (1994) tried different parallel GAs for this problem. Two of the parallel GAs were coarse-grained (with four demes) and the other was a global GA that used 16 worker processes to evaluate the population. In the first of the coarse-grained GAs the demes executed in isolation, and in the second GA the demes communicated with some (unspecified) programmable schedule. This paper is very interesting because it contains two uncommon elements: first, the parallel GAs were implemented using Linda, a parallel coordination language that is usually regarded as being slow; second, the speedups are measured using the times it takes to find a solution of a certain fixed quality by the serial and the parallel GAs. Computing the speedup in this way gives a more significant measure of the performance of the parallel GAs than just comparing the times it takes to run for some number of generations.

8 Non-traditional GAs

Not all coarse-grained parallel GAs use the traditional generational GA like the one reviewed in section 2. Some non-traditional GAs have also been parallelized and some improvements has been reported over their serial versions. For example, Maresky (1994) used Davidor's ECO model (Davidor, 1991) in the nodes for his distributed algorithm and explored different migration schemes with varying complexity. The simplest scheme was to do no migration at all and just collect the results from each node at the end of a run. The more complex algorithms involved the migration of complete ECO demes. Several replacement policies were explored with only marginal benefits.

GENITOR is a **steady-state GA** where only a few individuals change in every generation. In the distributed version of GENITOR (Whitley & Starkweather, 1990), individuals migrate at fixed intervals to neighboring demes and replace the worst individuals in the target deme. A significant improvement over the serial version was reported in the original paper and in later work by Gordon and Whitley (1993).

There have also been efforts to parallelize **messy GAs** (Goldberg, Korb, & Deb, 1989). Messy GAs have two phases. In the *primordial* phase the initial population is created using a partial enumeration and it is reduced using tournament selection. Next, in the *juxtapositional* phase partial solutions found in the primordial phase are mixed together. The primordial phase dominates execution time and several data distribution strategies were tried by Merkle and Lamont (1993) extending the work reported previously by Dymek (1992). The results indicate that the quality of the solutions was not statistically different in most of the cases varying the distribution strategy, but there were gains in the execution time of the algorithm.

Later, a variation of the messy GA (a *fast* messy GA (Goldberg, Deb, Kargupta, & Harik, 1993)) was parallelized and used to search for optimal conformations of Met-Enkephalin molecules (Merkle, Gates, Lamont, & Pachter, 1993). In this problem, the algorithm showed good speedups for up to 32 processors, but with more than 32 nodes the speedup did not increase very fast.

Koza and Andre (1995) used a network of transputers to parallelize genetic programming. In their report they made an analysis of the computing and memory requirements of their algorithm and concluded that transputers were the most cost-effective solution to implement it. They did a coarse-grained implementation with 64 demes with 500 individuals each connected as a 2-D mesh. The authors experimented with different migration rates and reported super linear speedups in the computational effort (not the total time) over the panmictic version using moderate migration rates (around 8 percent).

9 Fine-grained parallel GAs

Recall that in the fine-grained model the population is divided into many small demes and that there is intensive communication between the demes. This massive communication provides a way to disseminate good solutions across the entire population. Again, selection and mating occur only within a deme.

Robertson (1987) parallelized the genetic algorithm of a classifier system on a Connection Machine 1. He parallelized selection of parents, the selection of classifiers to replace, mating, and crossover. The execution time of his implementation was independent of the number of classifiers (up to 16K, the number of processing elements in the CM-1).

The ASPARAGOS system was introduced by Gorges-Schleuter (Gorges-Schleuter, 1989; Gorges-Schleuter, 1989a) and Mühlenbein (Mühlenbein, 1989a; Mühlenbein, 1989b). ASPARAGOS used a population structure that looks like a ladder with the upper and lower ends tied together. ASPARAGOS was used to solve some difficult combinatorial optimization problems with great success. Later, a linear population structure was tried (Gorges-Schleuter, 1991) and different mating strategies were compared (Gorges-Schleuter, 1992).

ASPARAGOS uses a local hillclimbing algorithm to refine the individuals in its population. It is difficult to isolate the contribution of the spatial structure of the population on the quality of the search from the contribution from the hillclimber. However, all the empirical results gathered with ASPARAGOS showed that it is a very effective optimization tool.

Manderick and Spiessens (1989) implemented a fine-grained parallel GA with the population distributed on a 2-D grid. Selection and mating were only possible with neighbors and the authors observed that the performance of the algorithm degraded as the size of the deme increased. On the extreme, if the size of the neighborhood was big enough, this parallel GA was equivalent to a single panmictic population. The authors analyzed this algorithm theoretically in subsequent years (Spiessens & Manderick, 1990; Spiessens & Manderick, 1991). and their analysis showed that, for a deme size s and a string length l, the time complexity of the algorithm was $O(s+l)$ or $O(s(\log s)+l)$ time steps depending on the selection scheme used.

It is common to place the individuals of a fine-grained PGA in a 2-Dimensional grid because in many massively parallel computers the processing elements are connected using this topology. However, most of these computers also have a global router that can send messages to any processor in the network (at a higher cost) and other topologies can be simulated on top of the grid. There is a study by Schwehm (1992) that compared the effect of using different interconnection networks on fine-grained GAs. In this paper a graph partitioning problem was used as a benchmark to test different topologies simulated with a MasPar MP-1 computer. The topologies tested were a ring, a torus, a $16 \times 8 \times 8$ cube a $4 \times 4 \times 4 \times 4 \times 4$ cube, and a 10-D binary hypercube. The algorithm with the torus topology converged faster than the other algorithms, but there is no mention of the resulting quality.

Anderson and Ferris (1990) also tried different topologies and different replacement algorithms. They experimented with two rings, a hypercube, two meshes and an "island" topology where only one individual of each deme overlapped with other demes. They concluded that for the particular problem they were trying to solve (assembly line balancing) the ring and the "island" topologies were the best. Baluja compared two variations on a linear topology and a 2-D mesh (Baluja, 1992; Baluja, 1993) and found that the mesh gave the best results on almost all the problems tested.

Recently, Sarma and Jong (1996) analyzed the effects of the size and the shape of the neighborhood on the selection mechanism and found that the ratio of the radius of the neighborhood to the radius of the whole grid is a critical parameter that can be used to control the selection pressure over the whole population. Their analysis quantifies how the time of propagating a good solution across the whole population changes with different neighborhood sizes.

Of course, fine-grained parallel GAs have been used to solve some difficult application problems. A popular problem is two dimensional bin packing and satisfactory solutions have been found by Kroger et al. (Kröger, Schwenderling, & Vornberger, 1991; Kröger, Schwenderling, & Vornberger, 1992; Kröger, Schwenderling, & Vornberger, 1993). The job shop scheduling problem is also very popular in the GA literature and an example that applies fine-grained GAs to it is the work by Tamaki and Nishikawa (1992).

Shapiro and Navetta (1994) used a fine-grained GA algorithm to predict the secondary structure of RNA. They used the native X-Net topology of the MasPar-2 computer to define the neighborhood of each individual. The X-Net topology connects a processing element to eight other processors. Different logical topologies were tested with the GA: all eight nearest neighbors, four nearest neighbors, and all neighbors within a specified distance d. The results for $d > 2$ were the poorest and the four neighbor scheme consistently outperformed the topology with eight neighbors.

A few papers compare the performance of fine- and coarse-grained parallel GAs (Baluja, 1993; Gordon & Whitley, 1993). In these comparisons sometimes fine-grained GAs come ahead of the coarse-grained GAs, but sometimes it is just the opposite. The problem is that comparisons cannot be made in absolute terms, instead, we must make clear what do we want to minimize (e.g., wall clock time) or maximize (e.g., the quality of the solution) and make the comparison based on our criteria.

Gordon, Whitley, and Böhm (1992) showed that the critical path of a fine-grained algorithm is shorter than that of a coarse-grained GA. This means that if enough processors are available, massively parallel GAs would need less time to finish their execution regardless of the population size. However, it is important to note that this was a theoretical study that did not include considerations such as the communication bandwidth between processors or memory requirements. Gordon (1994) also studied the spatial locality of memory references in different models of parallel GAs. This analysis is useful in determining the most adequate computing platform for each model.

10 Hybrid parallel algorithms

A few researchers have tried to combine two of the methods to parallelize GAs, and this results in hybrid-parallel GAs. Some of these new hybrid algorithms add a new degree of complexity to the already complicated scene of parallel GAs, but other hybrids manage to keep the same complexity as one of their components.

When two methods of parallelizing GAs are combined they form a hierarchy and most of the hybrid parallel GAs are coarse-grained at the upper level. Some hybrids have a fine-

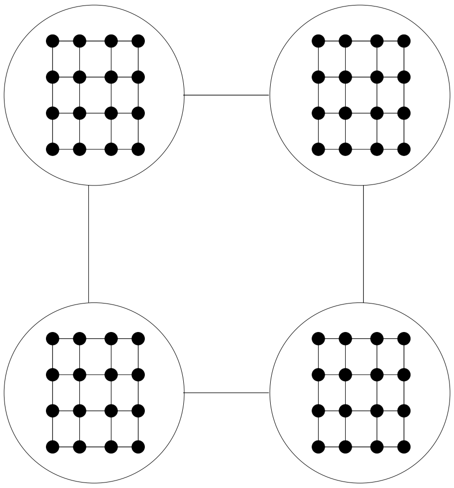

Figure 4: This hybrid GA combines a coarse-grained GA (at the high level) and a fine-grained GA (at the low level).

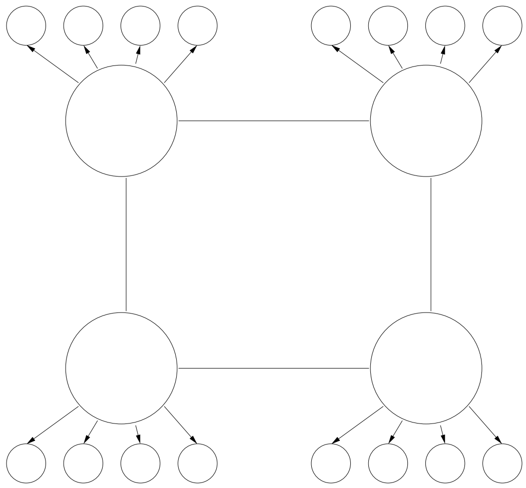

Figure 5: A schematic hybrid parallel GA. At the higher level this hybrid is a coarse-grained parallel GA where each node is a global parallel GA.

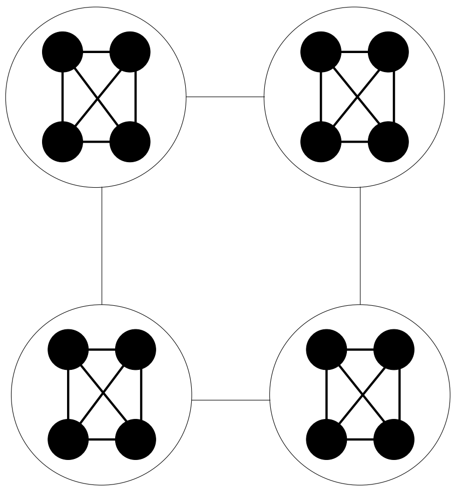

Figure 6: This hybrid uses coarse-grained GAs at both the high and the low levels. At the low level the migration rate is faster and the communications topology is much denser than at the high level.

grained GA at the lower level (see figure 4), for example Gruau (1994) invented a "mixed" parallel GA. In his algorithm the population of each deme is placed on a 2-D grid and the demes themselves are connected as a 2-D torus. Migration between neighboring demes occurs at regular intervals. Good results were reported for a novel neural network design and training application.

ASPARAGOS (Gorges-Schleuter, 1989b) was updated recently (Gorges-Schleuter, 1997) and its ladder structure was replaced with a ring, because the ring has a longer diameter and allows a better differentiation of the individuals. The new version (Asparagos96) maintains several subpopulations that are themselves structured as rings. Asparagos96 provides the option of migrating individuals across the subpopulation and in the experiments described by Gorges-Schleuter (1997) when a ring-structured population converges it receives the best individual of another population. After all the structured populations converge, Asparagos96 takes the best and second best individuals of each population and uses them as the initial population for a final run.

Lin, Goodman, and Punch (1997) also show a coarse-grained GA with spatially-structured subpopulations. Interestingly, they also used a ring topology —which is very sparse and has a long diameter— to connect the subpopulations, and each node is structured as a torus. Lin, Goodman, and Punch compared their hybrid against a fine-grained GA, two coarse-grained GAs with a ring topology (one used 5 demes and variable deme sizes, the other used a constant deme size of 50 individuals and different number of demes), and another coarse-grained GA with many demes connected as a torus. Using a job shop scheduling problem as a benchmark, they noticed that the fine-grained GA did not found solutions as good as the other methods, and that for the coarse-grained GAs increasing the number of demes improved the performance more than increasing the total population size. Their hybrid found better solutions overall.

Another way to hybridize a parallel GA is to use a form of global parallelization on each of the demes of a coarse-grained GA (see figure 5). Migration occurs between demes as in the coarse-grained algorithm, but the evaluation of the individuals is handled in parallel. This approach does not introduce new analytic problems and can be useful when working with complex applications with objective functions that need a considerable amount of computation time. Bianchini and Brown (1993) present an example of this method of hybridizing parallel GAs and showed experimentally that it can find a solution of the same quality of a global parallel GA or a conventional coarse-grained GA in less time.

Interestingly, a very similar concept was invented by Goldberg (1989) in the context of an object-oriented implementation of a "community model" parallel GA. In each "community" there are multiple houses where parents reproduce and the offspring are evaluated. Also, there are multiple communities and it is possible that individuals migrate to other places.

A third method of hybridizing parallel GAs is to use coarse-grained GAs at the high *and* at the low levels (see figure 6). The idea is to force panmictic mixing at the low level using a high migration rate and a dense topology and to use a low migration rate at the high level (Goldberg, 1996). This hybrid would also be equivalent in complexity to a coarse-grained GA if we consider the groups of panmictic subpopulations as a single deme. This method has not been implemented yet.

11 Recent advances

We have seen in previous sections that parallel GAs are effective in solving a number of difficult problems and that they can be implemented efficiently on different kinds of parallel computers. With all these positive results about parallel GAs it may be easy to overlook that there are still fundamental questions that remain unanswered.

The intent of this section is to summarize some important problems that affect the design of efficient parallel GAs and also to present a recent theory that examines the issue of deme sizes.

As we saw in section 2, the size of the population is probably the parameter that affects most the performance of the GA (Goldberg & Deb, 1991; Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997), so it is only natural to begin the study of parallel GAs with the size of the demes. Recently, Cantú-Paz and Goldberg (1997a) extended a theory that relates the population size in a serial GA with the expected quality of the solution to account for two bounding cases of coarse-grained GAs. The two bounding cases are a set of isolated demes and a set of fully connected demes. In the case of the connected demes the migration rate was set to the highest value possible.

The population size is not only the major factor in the quality of the solution that a GA is expected to find, it is also the major factor for determining the time that the GA needs to find the solution. Therefore, it is possible to use the deme sizing models to predict the computation times needed by a parallel GA and compare them with the time needed by a serial GA to reach a solution of the same quality. Cantú-Paz and Goldberg (1997b) combined their deme sizing models with a general power-law model for the communications time and predicted the expected parallel speedups for the two bounding cases that they studied.

There are two main conclusions from this theoretical analysis. First, the speedup that

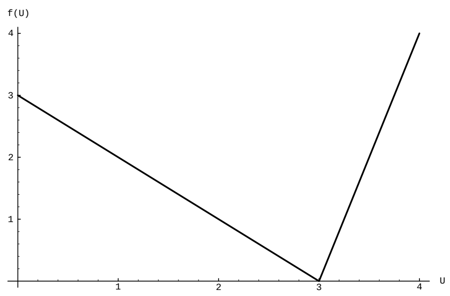

Figure 7: A deceptive 4-bit trap function of unity. The horizontal axis is the number of bits set to 1 and the vertical axis is the fitness value. The first test problem was constructed by concatenating 20 copies of this function and the second test problem is formed with 20 copies of a similar deceptive function of 8 bits.

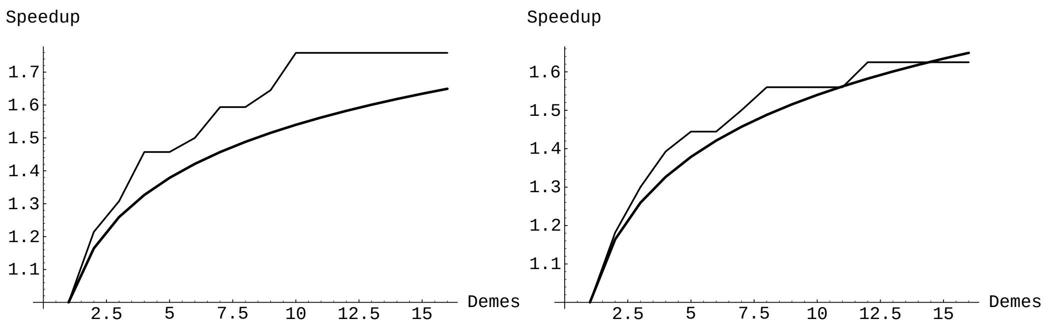

Figure 8: Projected and experimental speedups for test functions with 20 copies of a 4-bit trap (left) and 8-bit trap (right) functions using from 1 to 16 *isolated* demes. The thin lines show the experimental results and the thick lines are the theoretical predictions. The quality demanded was to find at least 16 copies correct.

is expected in the case where the demes execute in complete isolation is not very significant (see figures 7 and 8). Second, in the case where the demes communicate, there is an optimal number of demes (and an associated deme size) that maximizes the speedup (see figure 9).

The idealized bounding models can be extended in several directions, for example to consider smaller migration rates or more sparsely connected topologies, but the most important conclusion of this study is that there is a point after which adding more demes results in a slower algorithm. The existence of this optimal number of demes makes it necessary to use a different strategy to make parallel GAs faster, because adding more demes is not a viable option after some point. A good alternative is to hybridize the coarse-grained GA with one of the techniques reviewed in the previous section.

Parallel GAs are very complex and, of course,there are many problems that are still unresolved. A few examples are: (1) to determine the migration rate that makes distributed demes behave like a single panmictic population, (2) to determine an adequate commu-

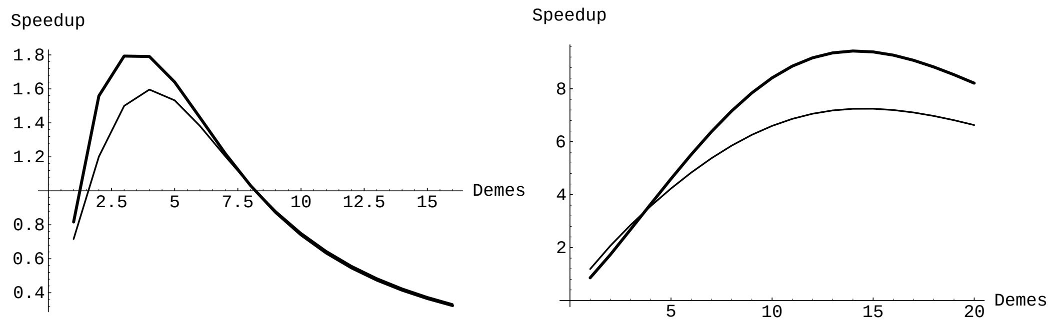

Figure 9: Projected and experimental speedups for test functions with 20 copies of a 4-bit trap (left) and 8-bit trap (right) functions using from 1 to 16 *fully connected* demes. The quality requirement was to find at least 16 copies correct. The thick lines are the theoretical predictions and the thin lines are the experimental results.

nications topology that permits the mixing of good solutions, but that does not result in excessive communication costs, and (3) to find if there is an optimal number of demes that maximizes reliability.

12 Summary and Conclusions

This paper reviewed some of the most representative publications on parallel genetic algorithms. The review started by classifying the work on this field into four categories: global parallelization, coarse- and fine-grained algorithms, and hybrid parallel GAs. We analyzed some of the most important contributions in each of these categories, trying to identify the issues that affect the design and the implementation of each class of parallel GAs on different types of parallel computers.

The research on parallel GAs is dominated by studies on coarse-grained algorithms and in consequence the paper focused on this type of implementation. The review of the literature on coarse-grained GAs revealed that there are several fundamental questions that remain unanswered many years after they were posed for the first time. The reason for this is that parallel GAs are very complex algorithms whose behavior is affected by many parameters. So it seems that the only way to achieve a deeper understanding of parallel GAs is to study individual facets independently, and we have seen that some of the most influential publications in parallel GAs concentrate on only one aspect (migration rates, communication topology, or deme size) ignoring or making simplifying assumptions about the others.

We also reviewed publications about global and fine-grained parallel GAs and realized that the combination of different parallelization strategies can result in faster algorithms. It is particularly important to consider the hybridization of parallel techniques in the light of recent results that predict the existence of an optimal number of demes.

As GAs are applied to bigger and more difficult search problems it becomes necessary to design faster algorithms that retain the capability of finding acceptable solutions. In this review we have seen numerous examples that show that parallel GAs are capable of combining speed and efficacy, and that we are reaching a deeper understanding that will allow us to utilize them better in the future.

Acknowledgments

I wish to think David E. Goldberg for his comments on an early version of this paper.

This study was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants number F49620-94-1-0103, F49620-95-1-0338, and F49620-97-1-0050. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

Erick Cantú-Paz was supported by a Fulbright-García Robles-CONACYT Fellowship.

References

Abramson, D., & Abela, J. (1992). A parallel genetic algorithm for solving the school timetabling problem. *Proceedings of the Fifteenth Australian Computer Science Conference (ACSC-15)*, *14*, 1–11.

Abramson, D., Mills, G., & Perkins, S. (1993). Parallelisation of a genetic algorithm for the computation of efficient train schedules. *Proceedings of the 1993 Parallel Computing and Transputers Conference*, 139–149.

Adamidis, P. (1994). *Review of parallel genetic algorithms bibliography* (Tech. Rep. Version 1). Thessaloniki, Greece: Aristotle University of Thessaloniki.

Anderson, E. J., & Ferris, M. C. (1990). A genetic algorithm for the assembly line balancing problem. In *Integer Programming and Combinatorial Optimization: Proceedings of a 1990 Conference Held at the University of Waterloo* (pp. 7–18). Waterloo, ON: University of Waterloo Press.

Baluja, S. (1992). *A massively distributed parallel genetic algorithm (mdpGA)* (Tech. Rep. No. CMU-CS-92-196R). Pittsburgh, PA: Carnegie Mellon University.

Baluja, S. (1993). Structure and performance of fine-grain parallelism in genetic search. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 155–162). San Mateo, CA: Morgan Kaufmann.

Belding, T. C. (1995). The distributed genetic algorithm revisited. In Eschelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 114–121). San Francisco, CA: Morgan Kaufmann.

Béssiere, P., & Talbi, E.-G. (1991, July). A parallel genetic algorithm for the graph partitioning problem. In *ACM Int. Conf. on Supercomputing ICS91*. Cologne, Germany.

Bethke, A. D. (1976). *Comparison of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity* (Tech. Rep. No. 197). Ann Arbor, MI: University of Michigan, Logic of Computers Group.

Bianchini, R., & Brown, C. M. (1993). Parallel genetic algorithms on distributed-memory architectures. In Atkins, S., & Wagner, A. S. (Eds.), *Transputer Research and Applications 6* (pp. 67–82). Amsterdam: IOS Press.

Branke, J., Andersen, H. C., & Schmeck, H. (1997, January). *Parallelising global selection in evolutionary algorithms.* Submitted to Journal of Parallel and Distributed Computing.

Braun, H. C. (1990). On solving travelling salesman problems by genetic algorithms. In Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature* (pp. 129–133). Berlin: Springer-Verlag.

Cantú-Paz, E. (1997). *Designing efficient master-slave parallel genetic algorithms* (IlliGAL Report No. 97004). Urbana, IL: University of Illinois at Urbana-Champaign.

Cantú-Paz, E., & Goldberg, D. E. (1997a). Modeling idealized bounding cases of parallel genetic algorithms. In Koza, J., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Iba, H., & Riolo, R. (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference* (pp. 353–361). San Francisco, CA: Morgan Kaufmann Publishers.

Cantú-Paz, E., & Goldberg, D. E. (1997b). Modeling speedups of idealized bounding cases of parallel genetic algorithms. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann Publishers.

Cantú-Paz, E., & Mejía-Olvera, M. (1994). Experimental results in distributed genetic algorithms. In *International Symposium on Applied Corporate Computing* (pp. 99–108). Monterrey, Mexico.

Chen, R.-J., Meyer, R. R., & Yackel, J. (1993). A genetic algorithm for diversity minimization and its parallel implementation. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 163–170). San Mateo, CA: Morgan Kaufmann.

Cohoon, J. P., Hegde, S. U., Martin, W. N., & Richards, D. (1987). Punctuated equilibria: A parallel genetic algorithm. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 148–154). Hillsdale, NJ: Lawrence Erlbaum Associates.

Cohoon, J. P., Martin, W. N., & Richards, D. S. (1991a). Genetic algorithms and punctuated equilibria in VLSI. In Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature* (pp. 134–144). Berlin: Springer-Verlag.

Cohoon, J. P., Martin, W. N., & Richards, D. S. (1991b). A multi-population genetic algorithm for solving the K-partition problem on hyper-cubes. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 244–248). San Mateo, CA: Morgan Kaufmann.

Cohoon, J. P., Martin, W. N., & Richards, D. S. (1991c). A multi-population genetic algorithm for solving the K-partition problem on hyper-cubes. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann Publishers.

Davidor, Y. (1991). A naturally occuring niche and species phenomenon: The model and first results. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 257–263). San Mateo, CA: Morgan Kaufmann.

Davis, M., Liu, L., & Elias, J. G. (1994). VLSI circuit synthesis using a parallel genetic algorithm. In Schaffer, J. D., Schwefel, H. P., & Kitano, H. (Eds.), *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 1 (pp. 104–109). Piscataway, NJ: IEEE Service Center.

Dymek, A. (1992). *An examination of hypercube implementations of genetic algorithms.* umt, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH.

Fogarty, T. C., & Huang, R. (1991). Implementing the genetic algorithm on transputer based parallel processing systems. *Parallel Problem Solving from Nature*, 145–149.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* New York: Addison-Wesley.

Goldberg, D. E. (1994). Genetic and evolutionary algorithms come of age. *Communications of the ACM*, *37*(3), 113–119.

Goldberg, D. E. (1996, June). Personal communication.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, *1*, 69–93. (Also TCGA Report 90007).

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 56–64). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530. (Also TCGA Report 89003).

Gordon, V. S. (1994). Locality in genetic algorithms. In Schaffer, J. D., Schwefel, H. P., & Kitano, H. (Eds.), *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 1 (pp. 428–432). Piscataway, NJ: IEEE Service Center.

Gordon, V. S., & Whitley, D. (1993). Serial and parallel genetic algorithms as function optimizers. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 177–183). San Mateo, CA: Morgan Kaufmann.

Gordon, V. S., Whitley, D., & Böhm, A. P. W. (1992). Dataflow parallelism in genetic algorithms. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, 2* (pp. 533–542). Amsterdam: Elsevier Science.

Gorges-Schleuter, M. (1989). ASPARAGOS: A population genetics approach to genetic algorithms. In Voigt, H.-M., Mühlenbein, H., & Schwefel, H.-P. (Eds.), *Evolution and Optimization '89* (pp. 86–94). Berlin: Akademie-Verlag.

Gorges-Schleuter, M. (1989a). ASPARAGOS: An asynchronous parallel genetic optimization strategy. *Proceedings of the Third International Conference on Genetic Algorithms*, 422–428.

Gorges-Schleuter, M. (1989b). ASPARAGOS: An asynchronous parallel genetic optimization strategy. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 422–428). San Mateo, CA: Morgan Kaufmann.

Gorges-Schleuter, M. (1991). Explicit parallelism of genetic algorithms through population structures. In Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature* (pp. 150–159). Berlin: Springer-Verlag.

Gorges-Schleuter, M. (1992). Comparison of local mating strategies in massively parallel genetic algorithms. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, 2* (pp. 553–562). Amsterdam: Elsevier Science.

Gorges-Schleuter, M. (1997). Asparagos96 and the traveling salesman problem. In Bäck, T. (Ed.), *Proceedings of the Fourth International Conference on Evolutionary Computation* (pp. 171–174). New York: IEEE Press.

Grefenstette, J. J. (1981). *Parallel adaptive algorithms for function optimization* (Tech. Rep. No. CS-81-19). Nashville, TN: Vanderbilt University, Computer Science Department.

Grosso, P. B. (1985). *Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model.* Unpublished doctoral dissertation, The University of Michigan. (University Microfilms No. 8520908).

Gruau, F. (1994). *Neural network synthesis using cellular encoding and the genetic algorithm.* Unpublished doctoral dissertation, L'Universite Claude Bernard-Lyon I.

Harik, G., Cantú-Paz, E., Goldberg, D., & Miller, B. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In Bäck, T. (Ed.), *Proceedings of the Fourth International Conference on Evolutionary Computation* (pp. 7–12). New York: IEEE Press.

Hauser, R., & Männer, R. (1994). Implementation of standard genetic algorithm on MIMD machines. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving fron Nature, PPSN III* (pp. 504–513). Berlin: Springer-Verlag.

Holland, J. H. (1959). A universal computer capable of executing an arbitrary number of sub-programs simultaneously. *Proceedings of the 1959 Eastern Joint Computer Conference*, 108–113.

Holland, J. H. (1960). Iterative circuit computers. In *Proceedings of the 1960 Western Joint Computer Conference* (pp. 259–265).

Koza, J. R., & Andre, D. (1995). *Parallel genetic programming on a network of transputers* (Tech. Rep. No. STAN-CS-TR-95-1542). Stanford, CA: Stanford University.

Kröger, B., Schwenderling, P., & Vornberger, O. (1991). Parallel genetic packing of rectangles. In Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature* (pp. 160–164). Berlin: Springer-Verlag.

Kröger, B., Schwenderling, P., & Vornberger, O. (1992). Massive parallel genetic packing. *Transputing in Numerical and Neural Network Applications*, 214–230.

Kröger, B., Schwenderling, P., & Vornberger, O. (1993). Parallel genetic packing on transputers. In Stender, J. (Ed.), *Parallel Genetic Algorithms: Theory and Applications* (pp. 151–185). Amsterdam: IOS Press.

Levine, D. (1994). *A parallel genetic algorithm for the set partitioning problem* (Tech. Rep. No. ANL-94/23). Argonne, IL: Argonne National Laboratory, Mathematics and Computer Science Division.

Li, T., & Mashford, J. (1990, 10-12October). A parallel genetic algorithm for quadratic assignment. In Ammar, R. A. (Ed.), *Proceedings of the ISMM International Conference. Parallel and Distributed Computing and Systems* (pp. 391–394). New York, NY: Acta Press, Anaheim, CA.

Lin, S.-C., Punch, W., & Goodman, E. (1994, October). Coarse-grain parallel genetic algorithms: Categorization and new approach. In *Sixth IEEE Symposium on Parallel and Distributed Processing.* Los Alamitos, CA: IEEE Computer Society Press.

Lin, S.-H., Goodman, E. D., & Punch, W. F. (1997, April). Investigating parallel genetic algorithms on job shop scheduling problem. In Angeline, P., Reynolds, R., J., M., & Eberhart, R. (Eds.), *Sixth International Conference on Evolutionary Programming* (pp. 383–393). Berlin: Springer Verlag.

Manderick, B., & Spiessens, P. (1989). Fine-grained parallel genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 428–433). San Mateo, CA: Morgan Kaufmann.

Maresky, J. G. (1994). *On effective communication in distributed genetic algorithms.* Master's thesis, Hebrew University of Jerusalem, Israel.

Marin, F. J., Trelles-Salazar, O., & Sandoval, F. (1994). Genetic algorithms on LAN-message passing architectures using PVM: Application to the routing problem. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving fron Nature, PPSN III* (pp. 534–543). Berlin: Springer-Verlag.

Merkle, L., Gates, G., Lamont, G., & Pachter, R. (1993). *Application of the parallel fast messy genetic algorithm to the protein folding problem* (Technical Report). Wright-Patterson AFB, Ohio: Air Force Institute of Technology.

Merkle, L. D., & Lamont, G. B. (1993). Comparison of parallel messy genetic algorithm data distribution strategies. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 191–198). San Mateo, CA: Morgan Kaufmann.

Mühlenbein, H. (1989a). Parallel genetic algorithms, population genetics, and combinatorial optimization. In Voigt, H.-M., Mühlenbein, H., & Schwefel, H.-P. (Eds.), *Evolution and Optimization '89* (pp. 79–85). Berlin: Akademie-Verlag.

Mühlenbein, H. (1989b). Parallel genetic algorithms, population genetics and combinatorial optimization. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 416–421). San Mateo, CA: Morgan Kaufmann.

Mühlenbein, H. (1991). Evolution in time and space-The parallel genetic algorithm. In Rawlins, G. J. E. (Ed.), *Foundations of Genetic Algorithms* (pp. 316–337). San Mateo, CA: Morgan Kaufmann.

Mühlenbein, H. (1992). Darwin's continent cycle theory and its simulation by the prisoner's dilemma. In Verela, F. J., & Bourgine, P. (Eds.), *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artifical Life* (pp. 236–244). Cambridge, MA: The MIT Press.

Mühlenbein, H., Schomisch, M., & Born, J. (1991). The parallel genetic algorithm as function optimizer. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth*

International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann Publishers.

Munetomo, M., Takai, Y., & Sato, Y. (1993). An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 649). San Mateo, CA: Morgan Kaufmann.

Muntean, T., & Talbi, E.-G. (1991, October, 7-9). A parallel genetic algorithm for process-processors mapping. In Durand, M., & Dabaghi, E. (Eds.), *Proceedings of the Second Symposium II. High Performance Computing* (pp. 71–82). Montpellier, France: F. Amsterdam, Amsterdam.

Neuhaus, P. (1991). Solving the mapping problem—Experiences with a genetic algorithm. In Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature* (pp. 170–175). Berlin: Springer-Verlag.

Pettey, C. B., Leuze, M. R., & Grefenstette, J. J. (1987a). A parallel genetic algorithm. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 155–161). Hillsdale, NJ: Lawrence Erlbaum Associates.

Pettey, C. C., Leuze, M., & Grefenstette, J. J. (1987b). Genetic algorithms on a hypercube multiprocessor. *Hypercube Multiprocessors 1987*, 333–341.

Pettey, C. C., & Leuze, M. R. (1989). A theoretical investigation of a parallel genetic algorithm. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 398–405). San Mateo, CA: Morgan Kaufmann.

Robertson, G. G. (1987). Parallel implementation of genetic algorithms in a classifier system. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 140–147). Hillsdale, NJ: Lawrence Erlbaum Associates.

Sarma, J., & Jong, K. D. (1996). An analysis of the effects of neighborhood size and shape on local selection algorithms. In *Parallel Problem Solving from Nature IV* (pp. 236–244). Berlin: Springer-Verlag.

Schwehm, M. (1992). Implementation of genetic algorithms on various interconnection networks. In Valero, M., Onate, E., Jane, M., Larriba, J. L., & Suarez, B. (Eds.), *Parallel Computing and Transputer Applications* (pp. 195–203). Amsterdam: IOS Press.

Seredynski, F. (1994). Dynamic mapping and load balancing with parallel genetic algorithms. In Schaffer, J. D., Schwefel, H. P., & Kitano, H. (Eds.), *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 2 (pp. 834–839). Piscataway, NJ: IEEE Service Center.

Shapiro, B., & Navetta, J. (1994). A massively parallel genetic algorithm for RNA secondary structure prediction. *The Journal of Supercomputing*, *8*, 195–207.

Spiessens, P., & Manderick, B. (1990). A genetic algorithm for massively parallel computers. In Eckmiller, R., Hartmann, G., & Hauske, G. (Eds.), *Parallel Processing in Neural Systems and Computers, Dusseldorf, Germany* (pp. 31–36). Amsterdam, Netherlands: North-Holland.

Spiessens, P., & Manderick, B. (1991). A massively parallel genetic algorithm: Implementation and first analysis. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the*

Fourth International Conference on Genetic Algorithms (pp. 279–286). San Mateo, CA: Morgan Kaufmann.

Starkweather, T., Whitley, D., & Mathias, K. (1991). Optimization using distributed genetic algorithms. In Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature* (pp. 176–185). Berlin: Springer-Verlag.

Talbi, E.-G., & Bessière, P. (1991, June). A parallel genetic algorithm for the graph partitioning problem. In *Proc. of the International Conference on Supercomputing.* Cologne.

Tamaki, H., & Nishikawa, Y. (1992). A paralleled genetic algorihtm based on a neighborhood model and its application to the jobshop scheduling. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, 2* (pp. 573–582). Amsterdam: Elsevier Science.

Tanese, R. (1987). Parallel genetic algorithm for a hypercube. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 177–183). Hillsdale, NJ: Lawrence Erlbaum Associates.

Tanese, R. (1989a). Distributed genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 434–439). San Mateo, CA: Morgan Kaufmann.

Tanese, R. (1989b). *Distributed genetic algorithms for function optimization.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Whitley, D., & Starkweather, T. (1990). *Genitor II: A distributed genetic algorithm.* To appear in Journal of Experimental and Theoretical Artificial Intelligence.