# 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. As such, there has been extensive research in this field. This survey 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 parallel GAs with multiple populations, the survey focuses on this type of algorithms. Also, the paper describes some of the most significant problems in modeling and designing multi-population parallel GAs and presents some recent advancements.

KEYWORDS: Parallel genetic algorithms, master-slave genetic algorithms, multiple demes, hierarchical genetic algorithms

## 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 [GOL 94]. GAs are generally able to find good solutions in reasonable amounts of time, but as they are applied to harder and bigger problems there is an increase in the time required to find adequate solutions. As a consequence, 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 collect, organize, and present some of the most relevant publications on parallel GAs. In order to organize the growing amount of literature in this field, the paper presents a categorization of the different types of parallel implementations of GAs. By far, the most popular parallel GAs consist in multiple populations that evolve separately most of the time and exchange individuals occasionally. This type of parallel GAs is called multi-deme, coarse-grained or distributed GAs, and this survey concentrates on this class of algorithm. 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 a particular problem, but the intention of this paper is not to enumerate all the instances 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 it also mentions a few of the application problems to show that parallel GAs are useful in the "real world" and are not just an academic curiosity.

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

## 2. Genetic Algorithms — A Quick Introduction

This section provides 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 may consult the book by Goldberg [GOL 89a] for more detailed background information on GAs.

Genetic algorithms are stochastic search algorithms based on principles of natural selection and recombination. They attempt to find the optimal solution to the prob-

lem at hand by manipulating a population of candidate solutions. The population is evaluated and the best solutions are selected to reproduce and mate to form the next generation. Over a number of generations, good traits dominate the population, resulting in an increase in the quality of the solutions.

The basic mechanism in GAs is Darwinian evolution : bad traits are eliminated from the population because they appear in individuals which do not survive the process of selection. The good traits survive and are mixed by recombination (mating) to form better individuals. Mutation also exists in GAs, but it is considered a secondary operator. Its function is to ensure that diversity is not lost in the population, so the GA can continue to explore.

The notion of 'good' traits is formalized with the concept of building blocks (BBs), which are string templates (schemata) that match a short portion of the individuals and act as a unit to influence the fitness of individuals. The prevailing theory suggests that GAs work by propagating BBs using selection and crossover. We follow Goldberg, Deb, and Thierens [GOL 93b], and restrict the notion of a building block to the shortest schemata that contribute to the global optimum. In this view, the juxtaposition of two BBs of order $k$ at a particular string does not lead to a BB of order $2k$, but instead to two separate BBs.

Often, the individuals are composed of a binary string of a fixed length, $l$, and thus GAs explore a search space formed by $2^l$ points. Initially, the population consists of points chosen randomly, unless there is a heuristic to generate good solutions for the domain. In the latter case, a portion of the population is still generated randomly to ensure that there is some diversity in the solutions.

The size of the population is important because it influences whether the GA can find good solutions and the time it takes to reach them [GOL 92, HAR 97]. If the population is too small, there might not be an adequate supply of BBs, and it will be difficult to identify good solutions. If the population is too big, the GA will waste computational resources processing unnecessary individuals. This balance between the quality of the solution and the time that a simple GA needs to find it also exists for parallel GAs, and later we shall see how it affects their design.

Each individual in the population has a fitness value. In general, the fitness is a payoff measure that depends on how well the individual solves the problem. In particular, GAs are often used as optimizers, and the fitness of an individual is the value of the objective function at the point represented by the binary string. Selection uses the fitness value to identify the individuals that will reproduce and mate to form the next generation.

Simple GAs use two operators loosely based on natural genetics to explore the search space : crossover and mutation. Crossover is the primary exploration mechanism in GAs. This operator takes two random individuals from those already selected to form the next generation and exchanges random substrings between them. As an example, consider strings $A_1$ and $A_2$ of length 8 :

$A_1 =$ `0 1 1 0 | 1 1 1 1`

$A_2 =$ `1 1 1 0 | 0 0 0 0`

There are $l - 1$ possible crossover points in strings of length $l$. In the example, a

single crossover point is chosen randomly as 4 (as indicated by the symbol | above). Exchanging substrings around the crossover point results in two new strings for the next generation :

$A'_1$ = 0 1 1 0 0 0 0 0

$A'_2$ = 1 1 1 0 1 1 1 1

The recombination operation can take many forms. The example above used 1-point crossover, but it is possible to use 2-point, n-point, or uniform crossover. More crossover points result in a more exploratory search, but also increase the chance of destroying long BBs.

Mutation is usually considered a secondary search operator. Its function is to restore diversity that may be lost from the repeated application of selection and crossover. This operator alters some random value within a string. For example, take string $A_1$ = 1 1 0 1 1 and assume that position 3 is chosen randomly to mutate. The new string would be $A'_1$ = 1 1 1 1 1. As in Nature, the probability of applying mutation is very low in GAs, but the probability of crossover is usually high.

There are several ways to stop a GA. One method is to stop after a predetermined number of generations or function evaluations. Another is to stop when the average quality of the population does not improve after some number of generations. Another common alternative is to halt the GA when all the individuals are identical, which can only occur when mutation is not used.

## 3. A Classification of Parallel GAs

The basic idea behind most parallel programs is to divide a task into chunks and to solve the chunks simultaneously using multiple processors. This divide-and-conquer approach can be applied to GAs in many different ways, and the literature contains many examples of successful parallel implementations. Some parallelization methods use a single population, while others divide the population into several relatively isolated subpopulations. 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 used in this review is similar to others [ADA 94, GOR 93, LIN 94], but it is extended to include one more category.

There are three main types of parallel GAs : (1) global single-population master-slave GAs, (2) single-population fine-grained, and (3) multiple-population coarse-grained GAs. In a master-slave GA there is a single panmictic population (just as in a simple GA), but the evaluation of fitness is distributed among several processors (see Figure 1). Since in this type of parallel GA, selection and crossover consider the entire population it is also known as global parallel GAs. Fine-grained parallel GAs are suited for massively parallel computers and consist of one spatially-structured population. Selection and mating are restricted to a small neighborhood, but neighborhoods overlap permitting some interaction among all the individuals (see Figure 2 for a schematic of this class of GAs). The ideal case is to have only one individual for every processing element available.

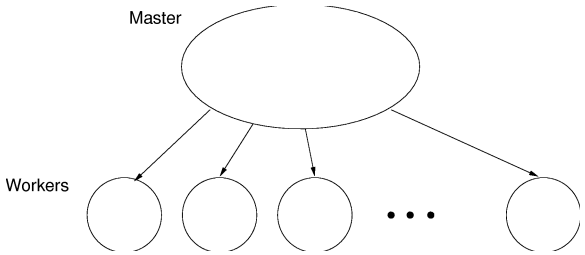



**Figure 1.** *A schematic of a master-slave parallel GA. The master stores the population, executes GA operations, and distributes individuals to the slaves. The slaves only evaluate the fitness of the individuals.*

Multiple-population (or multiple-deme) GAs are more sophisticated, as they consist on several subpopulations which exchange individuals occasionally (Figure 3 has a schematic). This exchange of individuals is called migration and, as we shall see in later sections, it is controlled by several parameters. Multiple-deme GAs are very popular, but also are the class of parallel GAs which is most difficult to understand, because the effects of migration are not fully understood. Multiple-deme parallel GAs introduce fundamental changes in the operation of the GA and have a different behavior than simple GAs.

Multiple-deme parallel GAs are known with different names. Sometimes they are known as "distributed" GAs, because they are usually implemented on distributed-memory MIMD computers. Since the computation to communication ratio is usually high, they are occasionally called coarse-grained GAs. Finally, multiple-deme GAs resemble the "island model" in Population Genetics which considers relatively isolated demes, so the parallel GAs are also known as "island" parallel GAs.

Since the size of the demes is smaller than the population used by a serial GA, we would expect that the parallel GA converges 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. Therefore, while it is true that smaller demes converge faster, 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 multi-deme parallel GAs, and allows us to make fair comparisons with serial GAs.

It is important to emphasize that while the master-slave parallelization method does not affect the behavior of the algorithm, the last two methods change the way the GA works. For example, in master-slave parallel GAs, selection takes into account all the population, but in the other two parallel GAs, selection only considers a subset of individuals. Also, in the master-slave any two individuals in the population can mate (i.e., there is random mating), but in the other methods mating is restricted to a subset of individuals.

The final method to parallelize GAs combines multiple demes with master-slave or fine-grained GAs. We call this class of algorithms hierarchical parallel GAs, because at

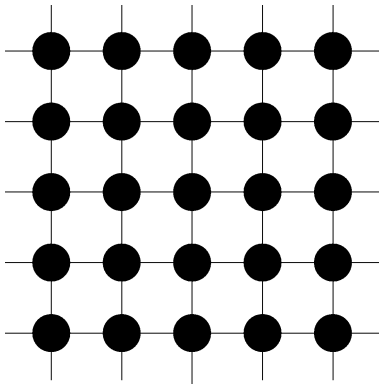


**Figure 2.** *A schematic of a fine-grained parallel GA. This class of parallel GAs has one spatially-distributed population, and it can be implemented very efficiently on massively parallel computers.*

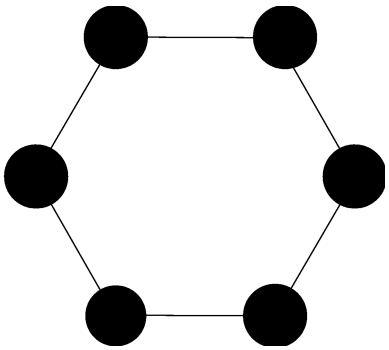


**Figure 3.** *A schematic of a multiple-population parallel GA. Each process is a simple GA, and there is (infrequent) communication between the populations.*

a higher level they are multiple-deme algorithms with single-population parallel GAs (either master-slave or fine-grained) at the lower level. A hierarchical parallel GAs combines the benefits of its components, and it promises better performance than any of them alone. Section 10 reviews the results of some implementations of hierarchical 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 [HOL 59, HOL 60]. 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 computer was the simulation of the evolution of natural populations.

The "iterative circuit computers", as they were called, were never built, but some of the hardware of the 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 more recent computers.

An early study of how to parallelize genetic algorithms was conducted by Bethke [BET 76]. 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 analyzed the efficiency of the both algorithms. His analysis showed that global parallel GAs may have an efficiency close to 100% in SIMD computers. He also analyzed gradient-based optimizers and identified some bottlenecks that limit their parallel efficiency.

Another early study on how to map genetic algorithms to existing computer architectures was made by Grefenstette [GRE 81]. He proposed four prototypes for parallel GAs ; the first three are variations of the master-slave scheme, and the fourth is a multiple-population GA. In the first prototype, the master processor stores the population, selects the parents of the next generation, 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 individual. This scheme eliminates the need to synchronize every generation and it can maintain a high level of processor utilization, even if the slave processors operate at different speeds. In the third prototype the population is stored in shared memory, which can be accessed by the slaves that evaluate the individuals and apply genetic operators independently of each other.

Grefenstette's fourth prototype is a multiple-population GA where the best individ-

uals of each processor are broadcast every generation to all the others. The complexity of multiple-deme 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 [PET 87b, PET 87a] and we review some of the results in Section 6. Grefenstette also hinted at the possibility of combining the fourth prototype with any of the other three, thereby creating a hierarchical parallel GA.

## 5. Master-Slave Parallelization

This section reviews the master-slave (or global) parallelization method. The algorithm uses a single population and the evaluation of the individuals and/or the application of genetic operators are done in parallel. As in the serial GA, each individual may compete and mate with any other (thus selection and mating are global). Global parallel GAs are usually implemented as master-slave programs, where the master stores the population and the slaves evaluate the fitness.

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 slave receives its subset of individuals to evaluate and when the slaves return the fitness values.

If the algorithm 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 master-slave GA has exactly the same properties as a simple GA, with speed being the only difference. However, it is also possible to implement an asynchronous master-slave GA where the algorithm does not stop to wait for any slow processors, but it does not work exactly like a simple GA. Most global parallel GA implementations are synchronous and the rest of the paper assumes that global parallel GAs carry out 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.

On a distributed-memory computer, the population can be stored in one processor. This "master" processor would be responsible for explicitly sending the individuals to the other processors (the "slaves") for evaluation, collecting the results, and applying the genetic operators to produce the next generation. The number of individuals assigned to any processor may 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 by using a dynamic scheduling

algorithm (e.g., guided self-scheduling).

Fogarty and Huang [FOG 91] attempted 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 communication between arbitrary nodes is handled by retransmitting messages. This causes an overhead in communications, and in an attempt to minimize it, Fogarty and Huang connected the transputers in different topologies. They concluded that the configuration of the network did not make a significant difference. They obtained reasonable speed-ups, but identified the fast-growing communication overhead as an impediment for further improvements in speed.

Abramson and Abela [ABR 92] implemented a GA on a shared-memory computer (an Encore Multimax with 16 processors) to search for efficient timetables for schools. They reported limited speed-ups, and blamed a few sections of serial code on the critical path of the program for the results. Later, Abramson, Mills, and Perkins [ABR 93] 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) speed-ups for up to 16 processors on the two computers, but the speed-ups degrade 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 [HAU 94]. They used three different parallel computers, but only obtained good speed-ups on a NERV multiprocessor (speed-up of 5 using 6 processors), that has a very low communications overhead. They explain the poor performance on the other systems they used (a SparcServer and a KSR1) on the inadequate scheduling of computation threads to processors by the system.

Another aspect of GAs that can be parallelized is the application of the genetic operators. Crossover and mutation can 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 also obstructs the parallelization of selection, mainly because several forms of selection need information about the entire population and thus require some communication. Recently, Branke [BRA 97] parallelized different types of global selection on a 2-D grid of processors and show that their algorithms are optimal for the topology they use (the algorithms take $O(\sqrt{n})$ time steps on a $\sqrt{n} \times \sqrt{n}$ grid).

In conclusion, master-slave parallel GAs are easy to implement and it can be a very efficient method of parallelization when the evaluation needs considerable computations. Besides, the method has the advantage of not altering the search behavior of the GA, so we can apply directly all the theory available for simple GAs.

## 6. Multiple-Deme Parallel GAs — The First Generation

The important characteristics of multiple-deme parallel GAs are the use of a few relatively large subpopulations and migration. Multiple-deme 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 parallel GAs with multiple populations was Grosso's dissertation [GRO 85]. His objective was to simulate the interaction of several parallel subcomponents of an evolving population. 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 others 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 confirms a long-held principle in Population Genetics : favorable traits spread faster when the demes are small than when the demes are large. However, he also observed that when the demes were isolated, the rapid rise in fitness stopped at a lower fitness value than with the large 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. However, 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 finds solutions of the same quality as the panmictic population.

In a similar parallel GA by Pettey, Leuze, and Grefenstette [PET 87a], a copy of the best individual found in each deme is 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 found solutions of the same quality as a sequential GA with a single large panmictic population. These observations prompted other questions that are still unresolved 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 to be able to answer these questions.

It is interesting 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 [TAN 87] proposed a parallel GA with the demes connected on a 4-D hyper-

cube topology. In Tanese's algorithm, migration occurred at fixed intervals between 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. Tanese carried out three sets of experiments. In the first, the interval between migrations was set to 5 generations, and the number of processors varied. In tests with two migration rates and varying the number of processors, the parallel GA found results of the same quality as the serial GA. 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 varied the mutation and crossover rates in each deme, attempting to find parameter values 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, Hedge, Martin, and Richards noted certain similarities between the evolution of solutions in the parallel GA and the theory of punctuated equilibria [COH 87]. This theory was proposed by Elredge and Gould to explain the 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 that changes on the environment can start a rapid evolutionary change. An important component of the environment of a population is its own composition, because individuals in a population must compete for resources with the other members. Therefore, the arrival of individuals from other populations can punctuate the equilibrium and trigger evolutionary changes. In their experiments with the parallel GA, Cohoon et al. noticed that there was relatively little change between migrations, but new solutions were found shortly after individuals were exchanged.

Cohoon et al. used a linear placement problem as a benchmark and experimented by 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 it 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 [COH 91a, COH 91b]. As 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 continued her work with a very exhaustive experimental study on the frequency of migrations and the number of individuals exchanged each time [TAN 89a]. She compared the performance of a serial GA against 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 multi-deme GA with no communication and $r$ demes of $n$ 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. With migration, the final average quality increased significantly, and 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 on the experiments and the conclusions derived from this study can be found in Tanese's dissertation [TAN 89b].

A recent paper by Belding [BEL 95] confirms Tanese's findings using Royal Road functions. Migrants were sent to a random destination, rather than to a neighbor in the hypercube topology, 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 multi-deme parallel GAs, we may hypothesize several reasons that make these algorithms so popular :

— Multiple-deme 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.

— There is relatively little extra effort needed to convert a serial GA into a multiple-deme GA. Most of the program of the serial GA remains the same and only a few subroutines need to be added to implement migration.

— Coarse-grain parallel computers are easily available, and even when they are not, it is easy to simulate one with a network of workstations or even on a single processor using free software (like MPI or PVM).

This section reviewed some of the papers that initiated the systematic research on parallel GAs with multiple populations, and that raised many of the important questions that this area still face today. The next section reviews more papers on multi-deme parallel GAs that show how the field matured and began to explore other aspects of 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. For example, parallel GAs are very promising in terms of the gains in performance. Also, parallel GAs are more complex than their serial counterparts. In particular, 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 the frequency of migrations.

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. This section reviews

some of that early theoretical work and experimental studies on migration and topologies. Also in this period, more researchers began to use multiple-population GAs to solve application problems, and this section ends with a brief review 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 is the work by Mühlenbein, Schomisch, and Born [MÜH 91b] who described a parallel GA that found the global optimum of several functions which are used as benchmarks for optimization algorithms. Later on, the functions used in this paper were adopted by other researchers as benchmarks for their own empirical work (e.g., [CHE 93, GOR 93]). Mühlenbein, Schomisch, and Born used a local optimizer in the algorithm, and although they designed an efficient and powerful program, it is not clear whether 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 [MÜH 89a, MÜH 89b, MÜH 92, MÜH 91a]. This view is shared by others who 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 with a direct counterpart in Nature (for example, 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 only to design faster search algorithms.

### 7.1. *Early Theoretical Work*

The conclusions of all the publications reviewed so far are based on experimental results, but after the first wave of publications describing successful implementations, a few theoretical studies appeared. A theoretical study of (parallel) GAs is necessary to achieve a deeper understanding of these algorithms, so that we can utilize them to their full potential.

Probably the first attempt to provide a theoretical foundation to the performance of a parallel GA was a paper by Pettey and Leuze [PET 89]. 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 under what conditions) a parallel GA can find a better solution than a serial GA. Starkweather, Whitley, and Mathias [STA 91] made two important observations regarding this question. First, relatively isolated demes are likely to converge to different solutions, and migration and recombination may combine partial solutions. Starkweather et al. speculated that because of this a parallel GA with low migration may find better solutions for separable functions. Their second observation was that if the recombination of partial solutions results in individuals with lower fitness (perhaps because the function is not separable), then the serial GA might have an advantage.

### 7.2. *Migration*

A sign of maturity of the field is that the research started to focus on particular aspects of parallel GAs. This section reviews some publications that explore alternative migration schemes to try to make parallel GAs more efficient.

In the majority of multi-deme parallel GAs, migration is synchronous which means that it occurs at predetermined constant intervals. Migration may also be asynchronous so that the demes communicate 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 is enabled until the population was close to converge.

Braun [BRA 90] used the same idea and presented an algorithm where migration occurred after the demes converged completely (the author used the term "degenerate") with the purpose of restoring diversity into the demes to prevent premature convergence to a low-quality solution. The same migration strategy was used later by Munetomo, Takai, and Sato [MUN 93], and recently Cantú-Paz and Goldberg [CAN 97b] presented theoretical models which predict the quality of the solutions when a fully connected topology is used.

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

A different approach to migration was developed by Marin, Trelles-Salazar, and Sandoval [MAR 94b]. They proposed a centralized scheme in which slave processors 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 (by any processor) and broadcasts them to the slaves. Experimental results on a network of workstations showed that near-linear speed-ups can be obtained with a small number of nodes (around six), and the authors claim that this approach can scale up to larger networks because the communication is 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 migration. For instance, a densely connected topology may promote a better mixing of individuals, but it also entails higher communication costs.

The general trend on multi-deme parallel GAs is to use static topologies that are specified at the beginning of the run and remain unchanged. Most implementations of parallel GAs with static topologies use the native topology of the computer available to the researchers. For example, implementations on hypercubes [COH 91a, TAN 89a, TAN 89b] and rings [GOR 93] 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 [CAN 94]. This study considered fully connected topologies, 4-D hypercubes, a $4 \times 4$ toroidal mesh, and unidirectional and bidirectional rings.

The other major choice is to use a dynamic topology. In this method, a deme is not restricted to communicate with some fixed set of demes, but instead the migrants are sent to demes that meet some criteria. The motivation behind dynamic topologies is to identify the demes where 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 [MUN 93] or a measure of the genotypic distance between the two populations [LIN 94] (or some representative individual of a population, like the best).

### 7.4. *Applications*

Many application projects using multiple-population parallel GAs have been published. This section mentions only a few representative examples.

The graph partitioning problem has been a popular application of multiple-deme GAs (see for example the work by Bessière and Talbi [BES 91], Cohoon, Martin, and Richards [COH 91c], and Talbi and Bessière [TAL 91]). The work of Levine [LEV 94] on the set partitioning 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 [LI 90] and Mühlenbein [MÜH 89b] found adequate 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 [NEU 91], Muntean and Talbi [MUN 91], and Seredynski [SER 94].

Another challenging application is the synthesis of VLSI circuits. Davis, Liu, and Elias [DAV 94] used different types of parallel GAs to search for solutions for this problem. Two of the parallel GAs use multiple populations (four demes), and the other is a global GA that uses 16 worker processes to evaluate the population. In the first of the multiple-deme GAs the demes are isolated, and in the second the demes communicate 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 (mis)regarded

as being slow ; second, the speed-ups 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 speed-up in this way gives a more significant measure of the performance of the parallel GAs than simply comparing the time it takes to run the GAs for a certain number of generations.

## 8. Non-Traditional GAs

Not all multiple-population parallel GAs use the traditional generational GA such as the one reviewed in Section 2. Some non-traditional GAs have also been parallelized and some improvements over their serial versions have been reported. For example, Maresky [MAR 94a] used Davidor's ECO model [DAV 91] 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 [WHI 90], individuals migrated at fixed intervals to neighbors, and replaced 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 [GOR 93].

There have also been efforts to parallelize messy GAs [GOL 89b]. 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. Since the primordial phase dominates the execution time, Merkle and Lamont tried several data distribution strategies to speed up this phase [MER 93b] extending previous work by Dymek [DYM 92]. The results indicate that there were no significant differences in the quality of the solutions for the different distribution strategies, but there were significant gains in the execution time of the algorithm.

Merkle, Gates, Lamont, and Pachter parallelized a fast messy GA [GOL 93a] to search for optimal conformations of Met-Enkephalin molecules [MER 93a]. In this problem, the algorithm showed good speed-ups for up to 32 processors, but with more than 32 nodes the speed-up did not increase very fast.

Koza and Andre [KOZ 95] 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. The 64 demes in their implementation were connected as a 2-D mesh and each deme had 500 individuals. The authors experimented with different migration rates, and reported super linear speed-ups in the computational effort (not the total time) using moderate migration rates (around 8%).

## 9. Fine-Grained Parallel GAs

Fine-grained parallel GAs have only one population, but it is has a spatial structure that limits the interactions between individuals. An individual can only compete and mate with its neighbors, but since the neighborhoods overlap good solutions may disseminate across the entire population.

Robertson [ROB 87] parallelized the genetic algorithm of a classifier system on a Connection Machine 1. He parallelized the 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).

ASPARAGOS was introduced by Gorges-Schleuter [Gor 89a, GOR 89b] and Mühlenbein [MÜH 89a, MÜH 89b]. ASPARAGOS used a population structure that resembles 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 used [Gor 91], and different mating strategies were compared [Gor 92b].

ASPARAGOS uses a local hillclimbing algorithm to improve the fitness of the individuals in its population. This makes it difficult to isolate the contribution of the spatial structure of the population to the search quality. However, all the empirical results show that ASPARAGOS is is a very effective optimization tool.

Manderick and Spiessens [MAN 89] implemented a fine-grained parallel GA with the population distributed on a 2-D grid. Selection and mating were only possible within a neighborhood, and the authors observed that the performance of the algorithm degraded as the size of the neighborhood 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 [SPI 90, SPI 91], 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.

Recently, Sarma and De Jong [SAR 96] 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 which determines the selection pressure over the whole population. Their analysis quantified how the time to propagate a good solution across the whole population changed with different neighborhood sizes.

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 with 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 may be simulated on top of the grid. Schwehm [SCH 92] compared the effect of using different structures on the population of fine-grained GAs. In this paper, a graph partitioning problem was used as a benchmark to test different population structures simulated on a MasPar MP-1 computer. The structures tested were a ring, a

torus, a $16 \times 8 \times 8$ cube a $4 \times 4 \times 4 \times 4 \times 4$ hypercube, and a 10-D binary hypercube. The algorithm with the torus structure converged faster than the other algorithms, but there is no mention of the resulting quality.

Andersson and Ferris [AND 90] also tried different population structures and replacement algorithms. They experimented with two rings, a hypercube, two meshes and an "island" structure 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" structures were the best. Baluja compared two variations on a linear structure and a 2-D mesh [BAL 92, BAL 93], and found that the mesh gave the best results on almost all the problems tested.

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 Kröger, Schwenderling, and Vornberger [KRÖ 91, KRÖ 92, KRÖ 93]. The job shop scheduling problem is also very popular in GA literature, and Yamaki and Nishikawa [YAM 92] successfully applied fine-grained GAs to find adequate solutions.

Shapiro and Naveta [SHA 94] 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 fine- and coarse-grained parallel GAs [BAL 93, GOR 93]. 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 particular criteria.

Gordon, Whitley, and Böhm [GOR 92a] showed that the critical path of a fine-grained algorithm is shorter than that of a multiple-population GA. This means that if enough processors were 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 communications bandwidth or memory requirements. Gordon [GOR 94] 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. Hierarchical Parallel Algorithms

A few researchers have tried to combine two of the methods to parallelize GAs, producing hierarchical parallel GAs. Some of these new hybrid algorithms add a new degree of complexity to the already complicated scene of parallel GAs, but other hy-

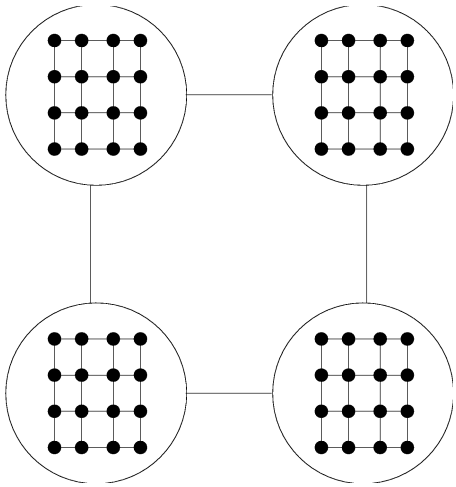


**Figure 4.** *This hierarchical GA combines a multi-deme GA (at the upper level) and a fine-grained GA (at the lower level).*

brids manage to keep the same complexity as one of their components.

When two methods of parallelizing GAs are combined they form a hierarchy. At the upper level most of the hybrid parallel GAs are multiple-population algorithms. Some hybrids have a fine-grained GA at the lower level (see Figure 4). For example Gruau [GRU 94] invented a "mixed" parallel GA. In his algorithm, the population of

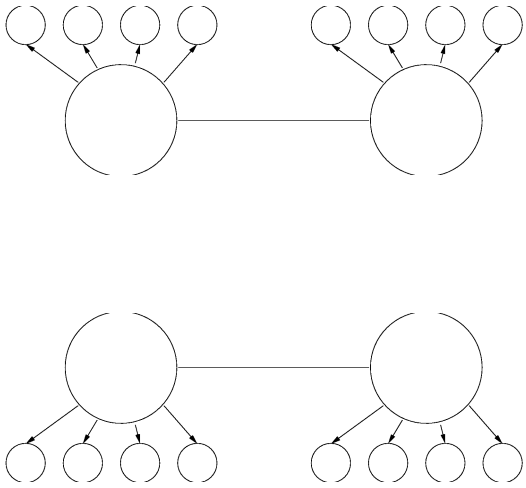


**Figure 5.** *A schematic of a hierarchical parallel GA. At the upper level this hybrid is a multi-deme parallel GA where each node is a master-slave GA.*

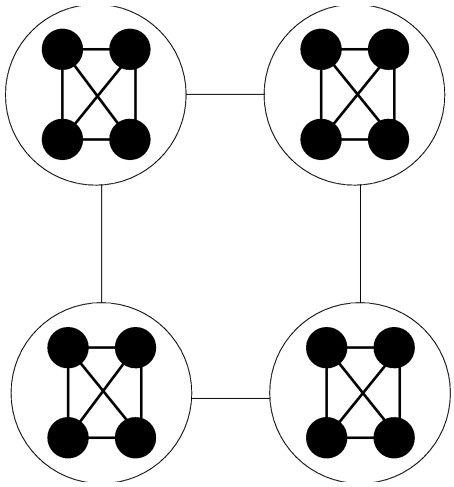


**Figure 6.** *This hybrid uses multi-deme GAs at both the upper and the lower levels. At the lower level the migration rate is faster and the communications topology is much denser than at the upper level.*

each deme was placed on a 2-D grid, and the demes themselves were connected as a 2-D torus. Migration between demes occurred at regular intervals, and good results were reported for a novel neural network design and training application.

ASPARAGOS was updated recently [GOR 97], and its ladder structure was replaced by 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. Migration across subpopulations is possible, and when a deme converges it receives the best individual from another deme. After all the 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 [LIN 97] also used a multi-population 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, but each deme was structured as a torus. The authors compared their hybrid against a simple GA, a fine-grained GA, two multiple-population GAs with a ring topology (one uses 5 demes and variable deme sizes, the other uses a constant deme size of 50 individuals and different number of demes), and another multi-deme GA with demes connected on a torus. Using a job shop scheduling problem as a benchmark, they noticed that the simple GA did not find solutions as good as the other methods, and that adding more demes to the multiple-population GAs improved the performance more than increasing the total population size. Their hybrid found better solutions overall.

Another type of hierarchical parallel GA uses a master-slave on each of the demes of a multi-population GA (see Figure 5). Migration occurs between demes, and the

evaluation of the individuals is handled in parallel. This approach does not introduce new analytic problems, and it can be useful when working with complex applications with objective functions that need a considerable amount of computation time. Bianchini and Brown [BIA 93] presented an example of this method of hybridizing parallel GAs, and showed that it can find a solution of the same quality of a master-slave parallel GA or a multi-deme GA in less time.

Interestingly, a very similar concept was invented by Goldberg [GOL 89a] 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 multi-deme GAs at both the upper and the lower levels (see Figure 6). The idea is to force panmictic mixing at the lower level by using a high migration rate and a dense topology, while a low migration rate is used at the high level [GOL 96]. The complexity of this hybrid would be equivalent to a multiple-population GA if we consider the groups of panmictic subpopulations as a single deme. This method has not been implemented yet.

Hierarchical implementations can reduce the execution time more than any of their components alone. For example, consider that in a particular domain the optimal speed-up of a master-slave GA is $Sp_{ms}$, and the optimal speed-up of a multiple-deme GA is $Sp_{md}$. The overall speed-up of a hierarchical GA that combines these two methods would be $Sp_{ms} \times Sp_{md}$.

## 11. Recent Advancements

This section summarizes some recent advancements in the theoretical study of parallel GAs. First, the section presents a result on master-slave GAs, and then there is a short presentation of a deme sizing theory for multiple-population GAs.

An important observation on master-slave GAs is that as more processors are used, the time to evaluate the fitness of the population decreases. But at the same time, the cost of sending the individuals to the slaves increases. This tradeoff between diminishing computation times and increasing communication times entails that there is an optimal number of slaves that minimizes the total execution time. A recent study [CAN 97a] concluded that the optimal is $S^* = \sqrt{\frac{nT_f}{T_c}}$, where $n$ is the population size, $T_f$ is the time it takes to do a single function evaluation, and $T_c$ is the communications time. The optimal speed-up is $0.5S^*$.

A natural starting point in the investigation of multiple-population GAs is the size of the demes, because the size of the population is probably the parameter that affects most the solution quality of the GA [GOL 91, GOL 92, HAR 97]. Recently, Cantú-Paz and Goldberg [CAN 97b] extended a simple GA population sizing model to account for two bounding cases of coarse-grained GAs. The two bounding cases were a set of isolated demes and a set of fully connected demes. In the case of the connected demes, the migration rate is set to the highest possible value.

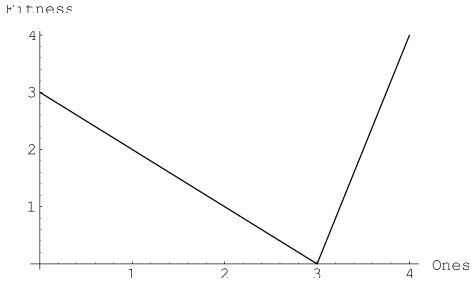


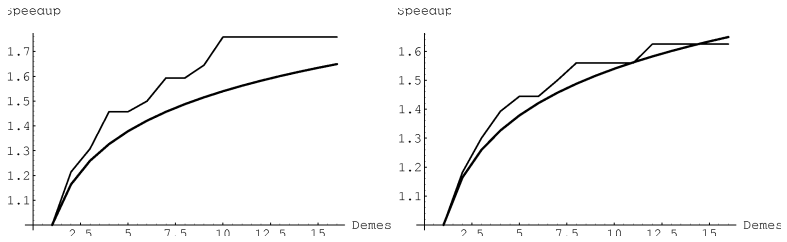

**Figure 7.** *A deceptive 4-bit trap function of unity. The horizontal axis is the number of bits set to 1 and the vertical axis is the fitness value. The difficulty of this function can be varied easily by using more bits in the basin of the deceptive optimum, or by reducing the fitness difference between the global and deceptive maxima. 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.*

The population size is also the major factor to determine the time that the GA needs to find the solution. Therefore, the deme sizing models may be used to predict the execution time of the parallel GA, and to compare it with the time needed by a serial GA to reach a solution of the same quality. Cantú-Paz and Goldberg [CAN 97c] integrated the deme sizing models with a model for the communications time, and predicted the expected parallel speed-ups for the two bounding cases.

There are three main conclusions from this theoretical analysis. First, the expected speed-up when the demes are isolated is not very significant (see Figures 7 and 8). Second, the speed-up is much better when the demes communicate. And finally, there is an optimal number of demes (and an associated deme size) that maximizes the speed-up (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. The fact that there is an optimal number of demes limits the processors that can be used to reduce the execution time. Using more than the optimal number of demes is wasteful, and would result in a slower algorithm. Hierarchical parallel GAs can use more processors effectively and reduce the execution time more than a pure multiple-deme GA.

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




(a) 4-bit trap (b) 8-bit trap

**Figure 8.** *Projected and experimental speed-ups for test functions with 20 copies of 4-bit and 8-bit trap 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.*

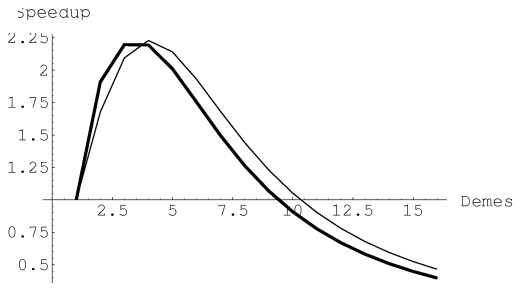
## 12. Summary and Conclusions

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 master-slave parallelization, fine-grained algorithms, multiple-deme, and hierarchical parallel GAs. Some of the most important contributions in each of these categories were analyzed, to try to identify the issues that affect the design and the implementation of each class of parallel GAs on existing parallel computers.

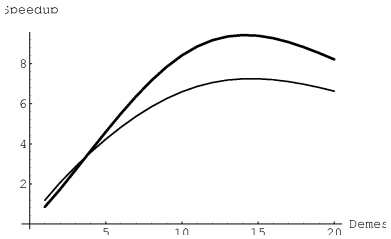
The research on parallel GAs is dominated by multiple-deme algorithms, and in consequence, this survey focused on them. The survey on multiple-population GAs revealed that there are several fundamental questions that remain unanswered many years after they were first identified. This class of parallel GAs is very complex, and its behavior is affected by many parameters. It seems that the only way to achieve a greater 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) either ignoring or making simplifying assumptions on the others.

We also reviewed publications on master-slave 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 which predict the existence of an optimal number of demes.

As GAs are applied to larger and more difficult search problems, it becomes necessary to design faster algorithms that retain the capability of finding acceptable solutions. This survey has presented numerous examples that show that parallel GAs are




(a) 4-bit trap




(b) 8-bit trap

**Figure 9.** *Projected and experimental speed-ups for test functions with 20 copies of 4-bit and 8-bit trap functions using from 1 to 16 fully connected demes. The quality requirement was to find at least 16 correct copies.*

capable of combining speed and efficacy, and that we are reaching a better understanding which should allow us to utilize them better in the future.

## Acknowledgments

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

This study was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant numbers 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

[ABR 92] ABRAMSON D., ABELA J., « A parallel genetic algorithm for solving the school timetabling problem ». In *Proceedings of the Fifteenth Australian Computer Science Conference (ACSC-15)*, vol. 14, p. 1–11, 1992.

[ABR 93] ABRAMSON D., MILLS G., PERKINS S., « Parallelisation of a genetic algorithm for the computation of efficient train schedules ». *Proceedings of the 1993 Parallel Computing and Transputers Conference*, p. 139–149, 1993.

[ADA 94] ADAMIDIS P., « Review of parallel genetic algorithms bibliography ». Tech. rep. version 1, Aristotle University of Thessaloniki, Thessaloniki, Greece, 1994.

[AND 90] ANDERSON E. J., FERRIS M. C., « 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*, p. 7–18, Waterloo, ON, 1990. University of Waterloo Press.

[BAL 92] BALUJA S., « A massively distributed parallel genetic algorithm (mdpGA) ». Tech. Rep. No. CMU-CS-92-196R, Carnegie Mellon University, Pittsburgh, PA, 1992.

[BAL 93] BALUJA S., « Structure and Performance of Fine-Grain Parallelism in Genetic Search ». In FORREST S., Ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 155–162, Morgan Kaufmann (San Mateo, CA), 1993.

[BEL 95] BELDING T. C., « The distributed genetic algorithm revisited ». In ESCHELMAN L., Ed., *Proceedings of the Sixth International Conference on Genetic Algorithms*, p. 114–121, Morgan Kaufmann (San Francisco, CA), 1995.

[BÉS 91] BÉSSIERE P., TALBI E.-G., « A parallel genetic algorithm for the graph partitioning problem ». In *ACM Int. Conf. on Supercomputing ICS91*, Cologne, Germany, July 1991.

[BET 76] BETHKE A. D., « Comparison of Genetic Algorithms and Gradient-Based Optimizers on Parallel Processors : Efficiency of Use of Processing Capacity ». Tech. Rep. No. 197, University of Michigan, Logic of Computers Group, Ann Arbor, MI, 1976.

[BIA 93] BIANCHINI R., BROWN C. M., « Parallel genetic algorithms on distributed-memory architectures ». In ATKINS S., WAGNER A. S., Eds., *Transputer Research and Applications 6*, p. 67–82, IOS Press (Amsterdam), 1993

[BRA 90] BRAUN H. C., « On Solving Travelling salesman Problems by Genetic Algorithms ». In SCHWEFEL H.-P., MANNER R., Eds., *Parallel Problem Solving from Nature*, p. 129–133, Springer-Verlag (Berlin), 1990.

[BRA 97] BRANKE J., ANDERSEN H. C., SCHMECK H., « Parallelising Global Selection in Evolutionary Algorithms ». Submitted to Journal of Parallel and Distributed Computing, January 1997.

[CAN 94] CANTÚ-PAZ E., MEJÍA-OLVERA M., « Experimental Results in Distributed Genetic Algorithms ». In *International Symposium on Applied Corporate Computing*, p. 99–108, Monterrey, Mexico, 1994.

[CAN 97a] CANTÚ-PAZ E., « Designing efficient master-slave parallel genetic algorithms ». IlliGAL Report No. 97004, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1997.

[CAN 97b] CANTÚ-PAZ E., GOLDBERG D. E., « 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*, Morgan Kaufmann (San Francisco, CA), 1997.

[CAN 97c] CANTÚ-PAZ E., GOLDBERG D. E., « Predicting speedups of idealized bounding cases of parallel genetic algorithms ». In BÄCK T., Ed., *Proceedings of the Seventh International Conference on Genetic Algorithms*, p. 113–121, Morgan Kaufmann (San Francisco), 1997.

[CHE 93] CHEN R.-J., MEYER R. R., YACKEL J., « A Genetic Algorithm for Diversity Minimization and Its Parallel Implementation ». In FORREST S., Ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 163–170, Morgan Kaufmann (San Mateo, CA), 1993.

[COH 87] COHOON J. P., HEGDE S. U., MARTIN W. N., RICHARDS D., « Punctuated equilibria : A parallel genetic algorithm ». In GREFENSTETTE J. J., Ed., *Proceedings of the Second International Conference on Genetic Algorithms*, p. 148–154, Lawrence Erlbaum Associates (Hillsdale, NJ), 1987.

[COH 91a] COHOON J. P., MARTIN W. N., RICHARDS D. S., « Genetic Algorithms and punctuated Equilibria in VLSI ». In SCHWEFEL H.-P., MANNER R., Eds., *Parallel Problem Solving from Nature*, p. 134–144, Springer-Verlag (Berlin), 1991.

[COH 91b] COHOON J. P., MARTIN W. N., RICHARDS D. S., « 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*, p. 244–248, Morgan Kaufmann (San Mateo, CA), 1991.

[COH 91c] COHOON J. P., MARTIN W. N., RICHARDS D. S., « 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*, Morgan Kaufmann (San Mateo, CA), 1991.

[DAV 91] DAVIDOR Y., « 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*, p. 257–263, Morgan Kaufmann (San Mateo, CA), 1991.

[DAV 94] DAVIS M., LIU L., ELIAS J. G., « VLSI circuit synthesis using a parallel genetic algorithm ». In *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. 1, p. 104–109, IEEE Press (Piscataway, NJ), 1994.

[DYM 92] DYMEK A., « An examination of hypercube implementations of genetic algorithms ». umt, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, 1992.

[FOG 91] FOGARTY T. C., HUANG R., « Implementing the genetic Algorithm on Transputer Based Parallel Processing Systems ». *Parallel Problem Solving from Nature*, p. 145–149, 1991.

[GOL 89a] GOLDBERG D. E., *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.

[GOL 89b] GOLDBERG D. E., KORB B., DEB K., « Messy genetic algorithms : Motivation, analysis, and first results ». *Complex Systems*, vol. 3, n° 5, p. 493–530, 1989. (Also TCGA Report 89003).

[GOL 91] GOLDBERG D. E., DEB K., « A comparative analysis of selection schemes used in genetic algorithms ». *Foundations of Genetic Algorithms*, vol. 1, p. 69–93, 1991. (Also TCGA Report 90007).

[GOL 92] GOLDBERG D. E., DEB K., CLARK J. H., « Genetic algorithms, noise, and the sizing of populations ». *Complex Systems*, vol. 6, p. 333–362, 1992.

[GOL 93a] GOLDBERG D. E., DEB K., KARGUPTA H., HARIK G., « Rapid, accurate optimization of difficult problems using fast messy genetic algorithms ». In FORREST S., Ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 56–64, Morgan Kaufmann (San Mateo, CA), 1993.

[GOL 93b] GOLDBERG D. E., DEB K., THIERENS D., « Toward a better understanding of mixing in genetic algorithms ». *Journal of the Society of Instrument and Control Engineers*, vol. 32, n° 1, p. 10–16, 1993.

[GOL 94] GOLDBERG D. E., « Genetic and evolutionary algorithms come of age ». *Communications of the ACM*, vol. 37, n° 3, p. 113–119, 1994.

[GOL 96] GOLDBERG D. E., June 1996. Personal communication.

[Gor 89a] GORGES-SCHLEUTER M., ASPARAGOS : A population genetics approach to genetic algorithms. In VOIGT H.-M., MÜHLENBEIN H., SCHWEFEL H.-P., Eds., *Evolution and Optimization '89*, p. 86–94. Akademie-Verlag (Berlin), 1989.

[GOR 89b] GORGES-SCHLEUTER M., « ASPARAGOS : An asynchronous parallel genetic optimization strategy ». In SCHAFFER J. D., Ed., *Proceedings of the Third International Conference on Genetic Algorithms*, p. 422–428, Morgan Kaufmann (San Mateo, CA), 1989.

[Gor 91] GORGES-SCHLEUTER M., « Explicit Parallelism of genetic Algorithms through Population Structures ». In SCHWEFEL H.-P., MÄNNER R., Eds., *Parallel Problem Solving from Nature*, p. 150–159, Springer-Verlag (Berlin), 1991.

[GOR 92a] GORDON V. S., WHITLEY D., BOHM A. P. W., « Dataflow parallelism in genetic algorithms ». In MÄNNER R., MANDERICK B., Eds., *Parallel Problem Solving from Nature, 2*, p. 533–542, Elsevier Science (Amsterdam), 1992.

[Gor 92b] GORGES-SCHLEUTER M., « Comparison of local mating strategies in massively parallel genetic algorithms ». In MÄNNER R., MANDERICK B., Eds., *Parallel Problem Solving from Nature, 2*, p. 553–562, Elsevier Science (Amsterdam), 1992.

[GOR 93] GORDON V. S., WHITLEY D., « Serial and Parallel Genetic Algorithms as Function Optimizers ». In FORREST S., Ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 177–183, Morgan Kaufmann (San Mateo, CA), 1993.

[GOR 94] GORDON V. S., « Locality in genetic algorithms ». In *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. 1, p. 428–432, IEEE Press (Piscataway, NJ), 1994.

[GOR 97] GORGES-SCHLEUTER M., « Asparagos96 and the Traveling Salesman Problem ». In BÄCK T., Ed., *Proceedings of the Fourth International Conference on Evolutionary Computation*, p. 171–174, IEEE Press (Piscataway, NJ), 1997.

[GRE 81] GREFENSTETTE J. J., « Parallel adaptive algorithms for function optimization ». Tech. Rep. No. CS-81-19, Vanderbilt University, Computer Science Department, Nashville, TN, 1981.

[GRO 85] GROSSO P. B., « *Computer simulations of genetic adaptation : Parallel subcomponent interaction in a multilocus model* ». Unpublished doctoral dissertation, The University of Michigan, 1985. (University Microfilms No. 8520908).

[GRU 94] GRUAU F., « *Neural network synthesis using cellular encoding and the genetic algorithm* ». Unpublished doctoral dissertation, L'Universite Claude Bernard-Lyon I, 1994.

[HAR 97] HARIK G., CANTÚ-PAZ E., GOLDBERG D. E., MILLER B. L., « The gambler's ruin problem, genetic algorithms, and the sizing of populations ». In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*, p. 7–12, IEEE Press (Piscataway, NJ), 1997.

[HAU 94] HAUSER R., MANNER R., « Implementation of standard genetic algorithm on MIMD machines ». In DAVIDOR Y., SCHWEFEL H.-P., MÄNNER R., Eds., *Parallel Problem Solving fron Nature, PPSN III*, p. 504–513, Springer-Verlag (Berlin), 1994.

[HOL 59] HOLLAND J. H., « A universal computer capable of executing an arbitrary number of sub-programs simultaneously ». *Proceedings of the 1959 Eastern Joint Computer Conference*, p. 108–113, 1959.

[HOL 60] HOLLAND J. H., « Iterative circuit computers ». In *Proceedings of the 1960 Western Joint Computer Conference*, p. 259–265, 1960.

[KOZ 95] KOZA J. R., ANDRE D., « Parallel genetic programming on a network of transputers ». Tech. Rep. No. STAN-CS-TR-95-1542, Stanford University, Stanford, CA, 1995.

[KRÖ 91] KROGER B., SCHWENDERLING P., VORNBERGER O., « Parallel genetic packing of rectangles ». In SCHWEFEL H.-P., MANNER R., Eds., *Parallel Problem Solving from Nature*, p. 160–164, Springer-Verlag (Berlin), 1991.

[KRÖ 92] KRÖGER B., SCHWENDERLING P., VORNBERGER O., « Massive parallel genetic packing ». *Transputing in Numerical and Neural Network Applications*, p. 214–230, 1992.

[KRÖ 93] KRÖGER B., SCHWENDERLING P., VORNBERGER O., Parallel genetic packing on transputers. In STENDER J., Ed., *Parallel Genetic Algorithms : Theory and Applications*, p. 151–185. IOS Press, Amsterdam, 1993.

[LEV 94] LEVINE D., « A parallel genetic algorithm for the set partitioning problem ». Tech. Rep. No. ANL-94/23, Argonne National Laboratory, Mathematics and Computer Science Division, Argonne, IL, 1994.

[LI 90] LI T., MASHFORD J., « A parallel genetic algorithm for quadratic assignment ». In AMMAR R. A., Ed., *Proceedings of the ISMM International Conference. Parallel and Distributed Computing and Systems*, p. 391–394, Acta Press (Anaheim, CA), 1990.

[LIN 94] LIN S.-C., PUNCH W., GOODMAN E., « Coarse-Grain Parallel Genetic Algorithms : Categorization and New Approach ». In *Sixth IEEE Symposium on Parallel and Distributed Processing*, IEEE Computer Society Press (Los Alamitos, CA), October 1994.

[LIN 97] LIN S.-H., GOODMAN E. D., PUNCH W. F., « 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*, p. 383–393, Springer-Verlag (Berlin), April 1997.

[MAN 89] MANDERICK B., SPIESSENS P., « Fine-grained parallel genetic algorithms ». In SCHAFFER J. D., Ed., *Proceedings of the Third International Conference on Genetic Algorithms*, p. 428–433, Morgan Kaufmann (San Mateo, CA), 1989.

[MAR 94a] MARESKY J. G., « On effective communication in distributed genetic algorithms ». Master's thesis, Hebrew University of Jerusalem, Israel, 1994.

[MAR 94b] MARIN F. J., TRELLES-SALAZAR O., SANDOVAL F., « Genetic algorithms on LAN-message passing architectures using PVM : Application to the routing problem ». In DAVIDOR Y., SCHWEFEL H.-P., MANNER R., Eds., *Parallel Problem Solving fron Nature, PPSN III*, p. 534–543, Springer-Verlag (Berlin), 1994.

[MER 93a] MERKLE L., GATES G., LAMONT G., PACHTER R., « Application of the Parallel Fast Messy Genetic Algorithm to the Protein Folding Problem ». Rapport technique, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1993.

[MER 93b] MERKLE L. D., LAMONT G. B., « Comparison of parallel messy genetic algorithm data distribution strategies ». In FORREST S., Ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 191–198, Morgan Kaufmann (San Mateo, CA), 1993.

[MÜH 89a] MUHLENBEIN H., Parallel genetic algorithms, population genetics, and combinatorial optimization. In VOIGT H.-M., MÜHLENBEIN H., SCHWEFEL H.-P., Eds., *Evolution and Optimization '89*, p. 79–85. Akademie-Verlag (Berlin), 1989.

[MÜH 89b] MÜHLENBEIN H., « Parallel genetic algorithms, population genetics and combinatorial optimization ». In SCHAFFER J. D., Ed., *Proceedings of the Third International Conference on Genetic Algorithms*, p. 416–421, Morgan Kaufmann (San Mateo, CA), 1989.

[MÜH 91a] MUHLENBEIN H., « Evolution in time and space-The parallel genetic algorithm ». In RAWLINS G. J. E., Ed., *Foundations of Genetic Algorithms*, p. 316–337, Morgan Kaufmann (San Mateo, CA), 1991.

[MÜH 91b] MUHLENBEIN H., SCHOMISCH M. , BORN J., « The Parallel Genetic Algorithm as Function Optimizer ». In BELEW R. K., BOOKER L. B., Eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann (San Mateo, CA), 1991.

[MUH 92] MUHLENBEIN H., « 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*, p. 236–244, MIT Press (Cambridge, MA), 1992.

[MUN 91] MUNTEAN T., TALBI E.-G., « A Parallel Genetic Algorithm for process-processors mapping ». In DURAND M., DABAGHI E., Eds., *Proceedings of the Second Symposium II. High Performance Computing*, p. 71–82, Montpellier, France, October, 7-9 1991.

[MUN 93] MUNETOMO M., TAKAI Y., SATO Y., « An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms ». In FORREST S., Ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 649, Morgan Kaufmann (San Mateo, CA), 1993.

[NEU 91] NEUHAUS P., « Solving the mapping Problem—Experiences with a Genetic Algorithm ». In SCHWEFEL H.-P., MÄNNER R., Eds., *Parallel Problem Solving from Nature*, p. 170–175, Springer-Verlag (Berlin), 1991.

[PET 87a] PETTEY C. B., LEUZE M. R., GREFENSTETTE J. J., « A parallel genetic algorithm ». In GREFENSTETTE J. J., Ed., *Proceedings of the Second International Conference on Genetic Algorithms*, p. 155–161, Lawrence Erlbaum Associates (Hillsdale, NJ), 1987.

[PET 87b] PETTEY C. C., LEUZE M., GREFENSTETTE J. J., « Genetic algorithms on a hypercube multiprocessor ». *Hypercube Multiprocessors 1987*, p. 333–341, 1987.

[PET 89] PETTEY C. C., LEUZE M. R., « A theoretical investigation of a parallel genetic algorithm ». In SCHAFFER J. D., Ed., *Proceedings of the Third International Conference on Genetic Algorithms*, p. 398–405, Morgan Kaufmann (San Mateo, CA), 1989.

[ROB 87] ROBERTSON G. G., « Parallel implementation of genetic algorithms in a classifier system ». In GREFENSTETTE J. J., Ed., *Proceedings of the Second International Conference on Genetic Algorithms*, p. 140–147, Lawrence Erlbaum Associates (Hillsdale, NJ), 1987.

[SAR 96] SARMA J., JONG K. D., « An analysis of the effects of neighborhood size and shape on local selection algorithms ». In *Parallel Problem Solving from Nature IV*, p. 236–244, Springer-Verlag (Berlin), 1996.

[SCH 92] SCHWEHM M., « 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*, p. 195–203, IOS Press (Amsterdam), 1992.

[SER 94] SEREDYNSKI F., « Dynamic mapping and load balancing with parallel genetic algorithms ». In *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. 2, p. 834–839, IEEE Press (Piscataway, NJ), 1994.

[SHA 94] SHAPIRO B., NAVETTA J., « A massively parallel genetic algorithm for RNA secondary structure prediction ». *The Journal of Supercomputing*, vol. 8, p. 195–207, 1994.

[SPI 90] SPIESSENS P., MANDERICK B., « A genetic algorithm for massively parallel computers ». In ECKMILLER R., HARTMANN G., HAUSKE G., Eds., *Parallel Processing in Neural Systems and Computers, Dusseldorf, Germany*, p. 31–36, North Holland (Amsterdam), 1990.

[SPI 91] SPIESSENS P., MANDERICK B., « 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*, p. 279–286, Morgan Kaufmann (San Mateo, CA), 1991.

[STA 91] STARKWEATHER T., WHITLEY D. , MATHIAS K., « Optimization Using Distributed Genetic Algorithms ». In SCHWEFEL H.-P., MANNER R., Eds., *Parallel Problem Solving from Nature*, p. 176–185, Springer-Verlag (Berlin), 1991.

[TAL 91] TALBI E.-G., BESSIÈRE P., « A Parallel Genetic Algorithm for the Graph Partitioning Problem ». In *Proc. of the International Conference on Supercomputing*, Cologne, June 1991.

[TAM 92] TAMAKI H., NISHIKAWA Y., « 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*, p. 573–582, Elsevier Science (Amsterdam), 1992.

[TAN 87] TANESE R., « Parallel genetic algorithm for a hypercube ». In GREFENSTETTE J. J., Ed., *Proceedings of the Second International Conference on Genetic Algorithms*, p. 177–183, Lawrence Erlbaum Associates (Hillsdale, NJ), 1987.

[TAN 89a] TANESE R., « Distributed genetic algorithms ». In SCHAFFER J. D., Ed., *Proceedings of the Third International Conference on Genetic Algorithms*, p. 434–439, Morgan Kaufmann (San Mateo, CA), 1989.

[TAN 89b] TANESE R., « *Distributed genetic algorithms for function optimization* ». Unpublished doctoral dissertation, University of Michigan, Ann Arbor, 1989.

[WHI 90] WHITLEY D., STARKWEATHER T., « Genitor II : A distributed genetic algorithm ». To appear in Journal of Experimental and Theoretical Artificial Intelligence, 1990.