On the Scalability of Parallel Genetic Algorithms

Erick Cantú-Paz*
Center for Applied Scientific Computing
Lawrence Livermore
National Laboratory
P.O. Box 808, L-551
Livermore, CA 94551, USA
cantupaz@llnl.gov

David E. Goldberg
Department of General Engineering
University of Illinois
at Urbana-Champaign
Urbana, IL 61801, USA
deg@illigal.ge.uiuc.edu

Abstract

This paper examines the scalability of several types of parallel genetic algorithms (GAs). The objective is to determine the optimal number of processors that can be used by each type to minimize the execution time. The first part of the paper considers algorithms with a single population. The investigation focuses on an implementation where the population is distributed to several processors, but the results are applicable to more common master-slave implementations, where the population is entirely stored in a master processor and multiple slaves are used to evaluate the fitness. The second part of the paper deals with parallel GAs with multiple populations. It first considers a bounding case where the connectivity, the migration rate, and the frequency of migrations are set to their maximal values. Then, arbitrary regular topologies with lower migration rates are considered and the frequency of migrations is set to its lowest value. The investigation is mainly theoretical, but experimental evidence with an additively-decomposable function is included to illustrate the accuracy of the theory. In all cases, the calculations show that the optimal number of processors that minimizes the execution time is directly proportional to the square root of the population size and the fitness evaluation time. Since these two factors usually increase as the domain becomes more difficult, the results of the paper suggest that parallel GAs can integrate large numbers of processors and significantly reduce the execution time of many practical applications.

Keywords

Master-slave genetic algorithms, multiple demes, island model, deme size, topology, migration rate, bounding cases.

1 Introduction

Early proposals of parallel genetic algorithms (GAs) recognized two forms of parallelization that are still common today: multiple communicating populations, and single-population master-slave implementations (Bethke, 1976; Grefenstette, 1981). The two types of parallel GAs have been extensively used to reduce the execution time of a variety of applications.

The choice between the two types of parallel GAs is determined by several factors such as ease of use or implementation, and by their potential to reduce the execution time. In

*Work was conducted while the author was a student at the University of Illinois, Department of Computer Science.

general, single-population parallel GAs are the easiest to use and implement, because any available knowledge about configuring serial GAs can be applied directly to them, and their implementation as master-slave algorithms is not complicated. In contrast, parallel GAs with multiple populations are more difficult to use, because one must choose adequate values for several additional parameters that affect the efficiency and the quality of the solutions that the algorithms reach. Among other things, to use multiple-deme GAs, one must decide the number and the size of the populations (also called demes), the frequency of migration, and the number and destination of the migrants. This paper examines the scalability of both single and multiple-population parallel GAs. The goal of the paper is to determine the optimal number of processors that minimize the execution time of each type of parallel GA. Examination of single-population algorithms is straightforward, and for multiple-population GAs, the paper examines cases that use upper and lower bounds of some of their parameters. The results suggest that the optimal number of processors for the two the classes of algorithms is of the same order, and that it is proportional to the square root of the product of the population size and the function evaluation time. There are other classes of parallel GAs such as fine-grained (Gorges-Schleuter, 1989; Mühlenbein, 1989; Manderick and Spiessens, 1989) and hierarchical parallel GAs (Gruau, 1994; Lin et al., 1997), but these classes are outside of the scope of this paper.

The study is mainly theoretical, but experimental results with an additively-decomposable test function are included to illustrate the accuracy of the models of solution quality. Only one function was used in the tests reported here, because the models are based on the gambler's ruin model of population sizing (Harik et al., 1997), which has been verified extensively with multiple additively-decomposable functions of varying difficulty. This leads us to believe that the results described here hold for any function of this class. The remainder of this paper is organized as follows. The next section discusses single-population algorithms and calculates the optimal number of processors that minimize the execution time. Section 3 briefly reviews previous studies on bounding cases of multi-deme GAs, and it describes the similarities of the single distributed population algorithm with a bounding case of multi-population GAs. Section 4 summarizes a model that relates the size of the population of a serial GA with the quality of the solutions it reaches. This model is extended in the following sections to multiple populations that communicate after they converge (i.e., all the individuals are identical, not necessarily representing the optimal solution). The populations restart after migration, and each convergence-migration sequence is called an epoch. Section 5 describes how the deme size, the migration rate, and the degree of the connectivity graph affect the chance that the desired solution is reached after two epochs. Section 6 generalizes the modeling to multiple epochs. Section 7 integrates the modeling of the previous sections with an estimate of the critical number of epochs until all the demes converge to the same solution and no further improvement is possible. The result is an estimate of the long-term execution time, which is then optimized with respect to the number of processors. Finally, Section 8 summarizes the results of this paper and presents the conclusions of this study.

2 Single-Population Parallel GAs

Probably the easiest way to parallelize GAs is to distribute the evaluation of fitness among several slave processors while one master executes the GA operations (selection, crossover, and mutation). Master-slave GAs are important for several reasons: (1) they explore the search space in exactly the same manner as a serial GA, and therefore the existing design

guidelines for simple GAs are directly applicable; (2) they are very easy to implement, which makes them popular with practitioners; and (3) in many cases master-slave GAs result in significant improvements in performance.

An alternative to storing the entire population in one node is to distribute the population to several nodes. However, the algorithm has to be modified so that the distributed population behaves as a single panmictic unit. As one might expect, the form of the execution time of the algorithm with a distributed population is very similar to the time of a simple master-slave algorithm, so the conclusions of this section apply to the two types of single-population parallel GAs. Moreover, the calculations are also important on another front: the algorithm with a distributed panmictic population resembles a bounding case of multi-deme parallel GAs, and therefore the predictions shed some light on the expected performance of this class of algorithms. This issue is explored in Section 3 and focuses on the single distributed population algorithm.

2.1 Description and Analysis

Assume that n is the population size required to reach the desired quality in a particular problem. In Section 4 we shall discuss how to determine n, but for now assume that the population size is given. The population is partitioned into equally-sized parts and distributed to $\mathcal{P}$ processors. In every generation, all the nodes perform the four basic GA tasks on their fraction of the population: fitness evaluation, selection, crossover, and mutation. The evaluations of fitness and mutation are not an issue, because they can be performed on each fraction of the population independently. However, to ensure that the distributed population behaves as a panmictic one, the selection and crossover operations must be modified. We begin the presentation of the algorithm with the parallel implementation of crossover, because ensuring that every individual is considered exactly once is straightforward.

In a sequential GA, each selected individual forms part of exactly one mating pair per generation. The pairs are formed randomly, and any two individuals in the population may be chosen to mate. To maintain this property in the parallel GA, each of the $\mathcal{P}$ processors randomly divides its fraction of the population into $\mathcal{P}$ equally-sized parts and sends each part to a different processor. These communications take time $T_x = (\mathcal{P} - 1)T_c$, where T_c is the average time used to communicate with one processor. The processors incorporate the incoming individuals into their fractions of the population by replacing the individuals sent to a particular deme with the individuals received from it. Then, each processor randomly chooses pairs of mates and proceeds to exchange material between them. There are, of course, other ways to preserve the panmictic property in a distributed population (e.g., Braud and Vrain (1999)), but the calculations would be similar.

The required modifications to selection are specific to the selection method used. To illustrate the modifications we consider tournament selection only, but other types of selection may be used. Recall that tournament selection without replacement works by choosing non-overlapping random sets of s individuals from the population and then selecting the best individual from each set to serve as a parent for the next generation. There are n/s such sets, and each tournament produces one winner; therefore to generate all the parents for the next generation, it is necessary to repeat this procedure s times. Any individual in the population may participate in any given tournament, and the key idea in the parallel algorithm is to maintain this property.

The first step in the parallel selection consists of each processor randomly dividing its

fraction of the population into $\mathcal{P}$ equally-sized parts and sending a different part to every other processor. These communications take time $(\mathcal{P}-1)T_c$. The processors receive and incorporate the individuals into their fraction of the population. Then, each processor picks random non-overlapping sets of s individuals to compete in tournaments. Note that the individuals who participate in a given tournament may have been received from any processor or may have been already present in the processor. In any case, all the individuals in the population have the same chance to participate in a given tournament, just as they do in the serial case. To select all the parents for the next generation, the process has to be repeated s times, and therefore the total communication time is $T_s = s(\mathcal{P}-1)T_c$.

The execution time of one generation of the algorithm is the sum of communications and computation times. Our calculations ignore the time used in the GA computations (selection, crossover, and mutation), because we assume that they are much smaller than the time used to evaluate and communicate individuals. Under this assumption, the computation time per node is $\frac{nT_f}{\mathcal{P}}$, where T_f is the time used to evaluate the fitness of one individual.

Communication is used during both selection (T_s) and crossover (T_x). The general form of the time used in communication during selection is $T_s = \kappa_s(\mathcal{P}-1)T_c$, where κ_s is a constant that depends on the selection method. In the case of tournament selection, κ_s equals to the size of the tournament s. The communication time used during crossover is $T_x = (\mathcal{P}-1)T_c$, so we may write the total time used in communication as $(\kappa_s+1)(\mathcal{P}-1)T_c$. To simplify the calculations we define $\kappa = \kappa_s + 1$. Adding computation and communication times we obtain the total execution time per generation:

$$T_p = \frac{nT_f}{\mathcal{P}} + \kappa(\mathcal{P}-1)T_c \tag{1}$$

As more processors are used, the computation time decreases as desired, but the communication time increases. This tradeoff entails the existence of an optimal number of processors that minimizes the execution time. The optimum may be found by solving $\frac{\partial T_p}{\partial \mathcal{P}} = 0$ for $\mathcal{P}$ to obtain

$$\mathcal{P}^* = \sqrt{\frac{nT_f}{\kappa T_c}} \tag{2}$$

In the case of a traditional master-slave algorithm, there is only one communication event per generation and therefore $\kappa = 1$. This result agrees with a previous calculation reported elsewhere (Cantú-Paz, 1998b).

In many practical situations, the function evaluation time is much greater than the time of communication, $T_f \gg T_c$, and the population size required to reach an acceptable solution is very large. Under those conditions, the optimal number of processors can be quite large and we can expect near-linear speedups for a wide range of processors as can be seen in Figure 1.

3 Bounding Cases of Multi-population GAs

Consider the typical parallel GA with multiple populations. Essentially, it consists of multiple GAs doing their normal selection-recombination-mutation cycle, but occasionally the GAs exchange some individuals (see the work by Grefenstette (1981), Grosso (1985),

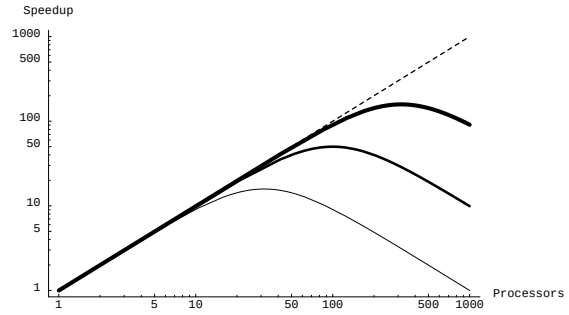


Figure 1: Theoretical speedups of single population parallel GAs varying the ratio of $\frac{T_f}{T_c}$. The thin line corresponds to $\frac{T_f}{T_c} = 1$, the intermediate to 10, and the thick to 100. The dotted line is the ideal (linear) speedup.

and Tanese (1987) for some early examples). These exchanges (migrations) are controlled by several parameters: the destination of the migrants, the frequency of migrations, and the number of individuals that are exchanged (the migration rate). It is difficult to predict the outcome of a run of multi-population GAs, because their parameters have non-linear effects on their efficiency and on the quality of the solutions that they reach. However, all of these parameters have minimum and maximum bounding values, and by studying parallel GAs that use the extreme values, we may gain some insight into the effects of the parameters.

Previous theoretical studies of bounding cases of multi-deme GAs focused on the number of demes and their sizes (Cantú-Paz and Goldberg, 1997a, 1997b). Those studies considered two configurations that bound the migration rate and the connectivity between the populations. In the first bounding case the demes are completely isolated, and it represents a lower bound on both the connectivity and the migration rate. The demes execute until they are composed of identical individuals, which is possible assuming that there is no mutation. The results of those studies suggest that the expected speedups in the isolated bounding case are only marginal, and therefore it should be avoided in practice.

In the second bounding case, each deme communicates with all the others, which is an upper bound on the connectivity, and the migration rate is set to its maximal value. Interdeme communications result in significant improvements in the quality of the solutions, which translate into important reductions of the deme size and the computational effort. However, using more processors requires additional communication and there is a tradeoff between increasing communication cost and decreasing computations. Cantú-Paz and Goldberg (1997b) showed how to use this tradeoff to compute the optimal number of processors that minimize the total execution time. Their investigation considered that communication between populations occurs after the populations converge, implying a lower bound on the frequency of migration.

The upper bound of the migration frequency is to exchange individuals every generation. In light of the significant reductions in computational effort caused by migration, it would be interesting to study the bounding case with the most intense migration possible. In fact, the distributed population described in the previous section resembles this bounding case: each processor executes a selection-crossover-mutation cycle, and there is an exchange

of individuals every generation. The topology of the communications is a fully connected graph (because processors communicate with every other), and dividing the population into $\mathcal{P}$ equal parts corresponds to the maximum possible migration rate.

However, the distributed population algorithm has *multiple* communications per generation and that is its major difference from the multi-population parallel GAs, where there is *at most one* migration event in a generation. In multi-population GAs, migrations occur after (or before) the selection-recombination-mutation sequence. This means that selection and recombination consider only a subset of the population, and the search may be biased toward some region of the search space. To our knowledge, there have been no studies of this bias, but it may be not very significant if migration occurs as often as every generation.

Another difference is that in multi-population algorithms the migrants and the individuals that they replace can be chosen in different ways. In particular, the (outgoing) migrants can be the best individuals in the population, or they may be chosen randomly. Likewise, when migrants arrive at a deme, they may replace individuals randomly or they may replace the worst. Each of these decisions affects the speed of convergence in some way, except when both the migrants and the individuals they replace are chosen randomly (Cantú-Paz, 1999a, 1999b), which is the case used in the algorithm described in the previous section.

The differences between a single distributed population and fully-connected populations that exchange the maximal number of individuals every generation are small, and therefore the calculations in the previous section serve as guidelines on how to choose the optimal number of populations for this bounding case.

4 The Gambler's Ruin Model

How does one determine the size of the populations? This section briefly reviews the gambler's ruin (GR) model that predicts the quality of solutions found by a GA with one population based on its size and the number of building blocks (BBs) present initially. Subsequent sections extend the model to consider multiple populations that communicate using arbitrary topologies.

To obtain a model of the quality of the solution of a GA, Harik et al. (1997) modeled the selection process in a GA as a biased one-dimensional random walk. The model considers one partition of order k, and it assumes that decisions are independent across partitions. We refer to the order-k schema that leads to the global optimum as the correct BB. Other schemata in the same partition—even if they have a high average fitness—are labeled as incorrect. In the GR model, the number of copies of the correct BB in a partition is represented by the position x of a particle on a one-dimensional space. The space is bounded with absorbing barriers at $x = 0$ and $x = n$, which represent ultimate convergence to the wrong and to the right solutions, respectively. The initial position of the particle x_0 is the expected number of copies of the correct BB in a randomly initialized population, and is equal to $x_0 = n/2^k$, where k is the order of the BB and n is the population size.

At each step of the random walk there is a probability p of obtaining one additional copy of the correct BB. This probability depends on the particular problem that the GA is facing, and it represents the probability of deciding correctly in a one-to-one competition between the best and the second best schemata of the partition. For functions composed by adding several uniformly-scaled subfunctions, p was computed by Goldberg et al. (1992) in their study of population sizing as

$$p = \Phi\left(\frac{d}{\sigma_{bb}\sqrt{2m'}}\right), \tag{3}$$

where Φ denotes the cumulative distribution function (CDF) of a normal distribution with a mean of zero and a standard distribution of one. d is the difference of the fitness contribution of the best and the second best schemata in the partition, $m' = m - 1$, m is the number of subfunctions, and σ_{bb}^2 is the average RMS variance of k-th order partitions.

A well-known result of random walks is the probability that a particle will eventually be captured by the absorbing barrier at $x = n$ (Feller, 1966):

$$P_{bb} = \frac{1 - \left(\frac{q}{p}\right)^{x_0}}{1 - \left(\frac{q}{p}\right)^{n}} \approx 1 - \left(\frac{q}{p}\right)^{x_0}, \tag{4}$$

where $q = 1 - p$. For increasing values of n, the denominator in Equation 4 approaches 1 very quickly, and therefore it may be ignored in the calculations.

We measure the quality of the solutions as the number of partitions that converged to the correct BB at the end of a run, and we denote the desired target quality as $\hat{Q}$. Using the assumption that partitions are independent, we can solve $P_{bb} = \frac{\hat{Q}}{m}$ for n and obtain the following population sizing equation:

$$n = \frac{2^k \ln(1 - \frac{\hat{Q}}{m})}{\ln \frac{q}{p}} \tag{5}$$

There are a number of assumptions in the gambler's ruin model. First, the GR model considers that decisions in a GA occur one at a time until all the n individuals in its population converge to the same value. In other words, in the model there is no explicit notion of generations. The model also assumes, conservatively, that all competitions occur between strings with the best and the second best schemata in a partition, and that the probability of deciding correctly remains constant during the run. Furthermore, the Goldberg et al. (1992) calculation of p implicitly assumed that the GA uses pairwise tournament selection (two strings compete), but adjustments for other selection schemes are possible, as Harik et al. (1997) showed in their paper.

The boundaries of the random walk are absorbing; this means that once a partition contains n copies of the correct BB it cannot lose one, and likewise, when the correct BB disappears from a partition there is no way of recovering it. This is related to another important assumption of the GR model: mutation and crossover do not create or destroy significant numbers of BBs. In the model, the only source of BBs is the random initialization of the population.

We must recognize that the GR model is a simplification, but experimental results with multiple additively decomposable functions of varying difficulty suggest that is it a reasonable one (Harik et al., 1997). The models in subsequent sections are largely based on the GR model, and therefore they inherit the assumptions, limitations, and applicability of the GR model.

In cases with multiple demes, success is defined when at least one of them reaches the desired target quality $\hat{Q}$. An equivalent success criterion is to require that the highest

quality found by any of the r demes equals the target quality. As a consequence, the quality required in each deme can be relaxed in the following way (Cantú-Paz and Goldberg, in press):

$$\hat{P} = \frac{\hat{Q}}{m} - \frac{\mu_{r:r}}{2\sqrt{m}} \approx \frac{\hat{Q}}{m} - \frac{\sqrt{\ln r}}{\sqrt{2m}}, \tag{6}$$

where $\mu_{r:r}$ is the expected value of the highest order statistic of r samples taken from a standard normal distribution with mean 0 and standard distribution of 1. The values of $\mu_{r:r}$ have been tabulated extensively (Harter, 1970), and the approximation $\mu_{r:r} = \sqrt{2 \ln r}$ used above was suggested by Beyer (1993). Note that $\hat{P}$ varies very slowly with respect to r, and that when $r = 1$, $\hat{P} = \hat{Q}$.

5 Multiple Demes and Regular Topologies

An important property of the connectivity graph between the demes is its *degree*, which is the number of neighbors of each deme. This paper assumes that all the demes have the same degree, and we denote it as δ. The degree completely determines the cost of communications, and as we shall see, it also influences the size of the demes and consequently the time of computations.

This section examines how the deme size, the migration rate, and the degree of the topology affect the probability that the parallel GA reaches the desired solution. The calculations consider that the populations communicate only after they have converged. After migration, the algorithm restarts. This represents a lower bound on the frequency of migrations. Similar algorithms were investigated empirically by Grosso (1985), Braun (1990), and Munetomo et al. (1993). Only the first two epochs of the algorithm are considered initially, because closed-form expressions may be derived easily. The next section extends the modeling to multiple epochs. Some of the results in this section have been presented elsewhere (Cantú-Paz and Goldberg, in press).

The modeling has several steps. First, we compute how many copies of the correct BB are necessary to reach the target quality per deme ($\hat{P}$, given by Equation 6). Next, the probability that a given configuration brings together the critical number of BBs is calculated. The success probability is then used to derive a deme sizing equation, which in turn is used to minimize the execution time.

The first step of the modeling is straightforward. To determine how many BBs $\widehat{x_1}$ are needed at the beginning of the second epoch to reach $\hat{P}$, we may use the solution of the gambler's ruin problem. Making

$$\hat{P} = 1 - \left(\frac{q}{p}\right)^{\widehat{x_1}},$$

and solving for $\widehat{x_1}$ results in

$$\widehat{x_1} = \frac{\ln(1 - \hat{P})}{\ln \frac{q}{p}} \tag{7}$$

The next step is to determine the probability that a deme receives at least $\widehat{x_1}$ BBs from its δ neighbors. The probability that one neighbor sends the right BB is the same probability

that it converged correctly in the first epoch, and is given by P_{bb}. Assuming that all the neighbors of a deme use the same migration rate ρ then at least $\hat{\delta} = \frac{\widehat{x_1}}{\rho n_d}$ neighbors must contribute the correct BB. Since the demes have evolved in isolation until this moment, the probability of receiving at least $\widehat{x_1}$ BBs has a binomial probability:

$$P_{x_1} = 1 - \sum_{i=0}^{\hat{\delta}-1} \binom{\delta}{i} P_{bb}^i (1 - P_{bb})^{\delta - i}, \tag{8}$$

which can be approximated as

$$P_{x_1} = 1 - \Phi \left(\frac{\hat{\delta} - \delta P_{bb}}{\sqrt{\delta P_{bb}(1 - P_{bb})}} \right) \tag{9}$$

Figure 2 shows the results of experiments that illustrate the accuracy of P_{x_1}. The test problem used in these experiments and in the remainder of the paper is $f = \sum_{i=1}^{20} f_4(u_{4i})$, where u_{4i} is the number of bits set to 1 in the 4-bit substring that starts at position $4i$, and f_4 is a fully deceptive trap function defined as:

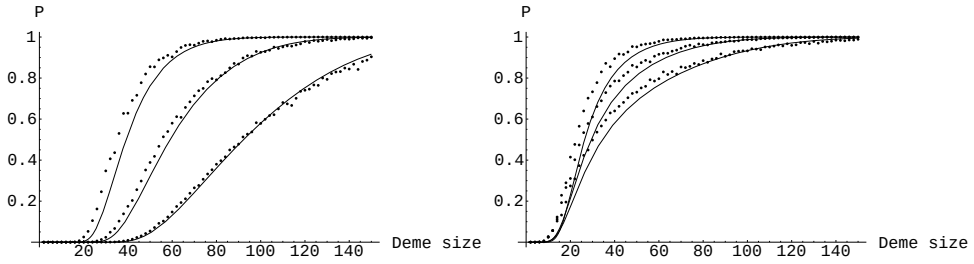
$$f_4(u) = \begin{cases} 3 - u & \text{if } u \in [0, 3], \\ 4 & \text{if } u = 4. \end{cases}$$

Fully deceptive trap functions are used in many studies of genetic algorithms, because their difficulty is well understood and it can be regulated easily (by changing the size of the deceptive basin or the fitness difference between the two peaks). The test function used in this paper has $m = 20$ copies of the trap function, the difference between the best and second best schemata is $d = 1$, the fitness variance in one partition is $\sigma_{bb}^2 = 1.215$, and therefore $p = 0.5585$. For the experiments, a simple generational GA is used in each deme with pairwise tournament selection without replacement, two-point crossover with probability 1, and no mutation. The graphs show the average of 100 independent runs.

With higher migration rates, fewer neighbors must contribute the correct BB, and therefore it is more likely that the deme receives the critical number of BBs and succeeds to find the solution. This observation agrees with other studies that concluded that the solution's quality improves with higher migration rates (Cantú-Paz, 1998b).

Note that even if a deme receives less than $\widehat{x_1}$ BBs, it may still reach the right solution, because the deme itself could have converged correctly in the first epoch, and it may contain enough BBs to converge correctly again. Also, a deme may start the second epoch with less than $\widehat{x_1}$ BBs and converge correctly sometimes. However, we ignore these two possibilities and conservatively assume that a deme does not converge to the right answer if it does not receive at least $\widehat{x_1}$ BBs from its neighbors. Under this assumption, P_{x_1} is the probability that at the end of the second iteration the deme will converge correctly.

There are different configurations that can bring together the critical number of BBs with the same probability. Configurations with large demes and few neighbors have the same chance of success than some configurations with smaller demes but with more neighbors (see Figure 3). This is the usual tradeoff between computation and communication: smaller demes require more neighbors to succeed. We would like to use the configuration that achieves the desired objective with the minimum cost.



(a) 10% migration (b) Maximal migration

Figure 2: Probability (Equation 9) of reaching the critical number of BBs required to find a solution with at least 16 out of 20 BBs ($\hat{Q} = 0.8$). The graphs show experimental results and the theoretical predictions using topologies with δ=1, 2, and 4 neighbors (from right to left in each graph) and different migration rates.

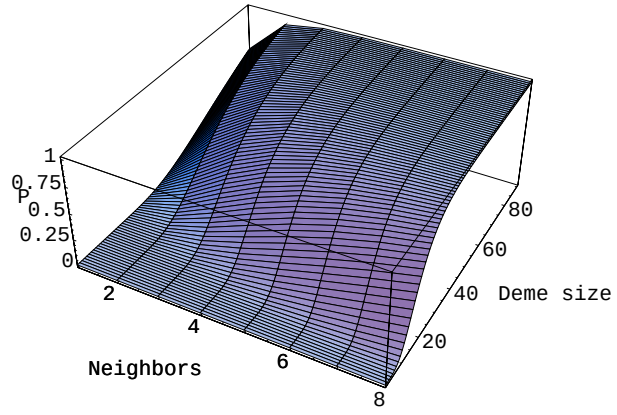


Figure 3: Plot of the probability of success with different configurations of deme sizes and number of neighbors (Equation 9). The fitness function is a 20-BB, 4-bit trap. The migration rate is $\rho = 0.10$.

The execution time of the parallel program is the sum of communication and computation times:

$$T_p = g n_d T_f + \delta T_c, \tag{10}$$

where g is the domain-dependent number of generations until convergence, n_d is the deme size, T_f is the time of one fitness evaluation, and T_c is the time required to communicate with one neighbor. T_c, T_f, and g can be easily determined empirically, but the required deme size depends on the degree of the topology, the migration rate, and the desired quality.

5.1 Finding the Deme Size

To find the deme size we need to make $P_{x_1} = \hat{P}$ and solve for n_d. First, the normal distribution of P_{x_1} has to be approximated as $\Phi(z) = (1 + \exp(-1.6z))^{-1}$ (Valenzuela-Rendón, 1989). With this approximation, P_{x_1} becomes

$$P_{x_1} = 1 - (1 + \exp(-1.6z))^{-1}, \tag{11}$$

where $z = \frac{\hat{\delta} - \delta P_{bb}}{\sqrt{\delta P_{bb}(1-P_{bb})}}$ is the normalized number of successes. We may bound z by considering that the variance is maximal when $P_{bb} = 0.5$, and thus it becomes $z \geq \frac{2}{\sqrt{\delta}}(\hat{\delta} - \delta P_{bb})$ (In the remainder we conservatively ignore the inequality.). Additionally, P_{bb} may be roughly approximated as $P_{bb} \approx \frac{cn}{2^k}$, where $c = 1 - q/p$. Substituting this form of P_{bb} and $\hat{\delta} = \frac{\widehat{x_1}}{\rho n_d}$ into the bound of z gives

$$z = \frac{2}{\sqrt{\delta}} \left(\frac{\widehat{x_1}}{\rho n_d} - \delta \frac{c n_d}{2^k} \right)$$

Making the approximate form of $P_{x_1} = \hat{P}$ and solving for z yields the ordinate, where the probability of success reaches the required value:

$$\hat{z} = 0.625 \ln \left(\frac{\hat{P}}{1 - \hat{P}} \right)$$

Making $z = \hat{z}$, solving for n_d, and simplifying terms gives the deme size:

$$n_d = \frac{2^{k-2}}{\sqrt{\delta}} \frac{\hat{z} + \sqrt{\hat{z}^2 + \frac{c\widehat{x_1}}{\rho 2^{k-2}}}}{c} \tag{12}$$

Observe that the deme size decreases with higher migration rates and as the number of neighbors increases, which is what we expected. For clarity, this deme-sizing equation may be rewritten in a more compact form by grouping all the domain-dependent constants into one (n_0) as follows:

$$n_d = \frac{n_0}{\sqrt{\delta}} \tag{13}$$

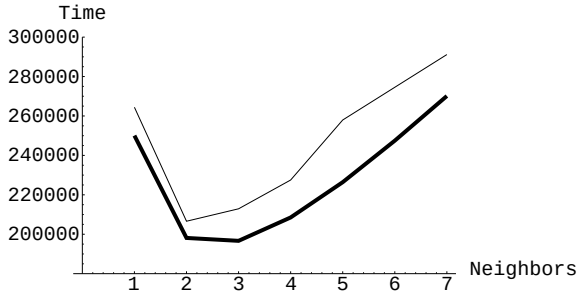


Figure 4: Comparison of theoretical (thick line) and experimental (thin line) execution times (in microseconds) of eight demes connected by topologies with different degrees.

Now, the total execution time as given by Equation 10 may be easily optimized with respect to δ by making $\frac{\partial T_p}{\partial \delta} = 0$ and solving for δ:

$$\delta^* = \left(\frac{g n_0 T_f}{2 T_c} \right)^{2/3}, \tag{14}$$

and the optimal deme size can be found by substituting δ^* in Equation 12.

Figure 4 compares the theoretical predictions of the execution time with experimental results on a network of eight IBM workstations. In this example, the fitness function consists of 20 copies of a 4-bit trap, and the objective is to find a solution with at least 16 partitions correct. The time to evaluate a single individual is $T_f = 51$ microseconds, the communications time is $T_c = 29$ ms, and the number of generations until convergence is $g = 50$. Figure 4 shows the average of 100 runs using pairwise tournament selection, two-point crossover with probability 1.0, and no mutation. The migration rate is $\rho = 0.1$.

5.2 Fully Connected Topologies Revisited

Sometimes the magnitude of the optimal degree will be greater than the number of demes, because it depends on the ratio of computation to communications, and this ratio may be arbitrarily large. In those situations, the topology that is closest to the optimal and that is realizable with the available demes would be a fully connected topology. This section revisits this bounding case and shows that, despite its poor scalability, the fully connected topology may be a good choice to reduce the execution time.

The calculation of the optimal degree may be used to find an expression for the optimal number of demes in a fully connected topology. At some deme count r the optimal degree will be $\delta^* = r - 1$, which is the degree of the fully connected topology. Solving for r gives the optimal number of fully-connected demes,

$$r^* = \delta^* + 1 \tag{15}$$

Figure 5 has an example with the same 20-BB, 4-bit trap function used before. The figure clearly shows that the optimal time varies very slowly when more than r^* demes are used, and that the optimal execution time of the fully connected case is very close to the optimal times of other topologies.

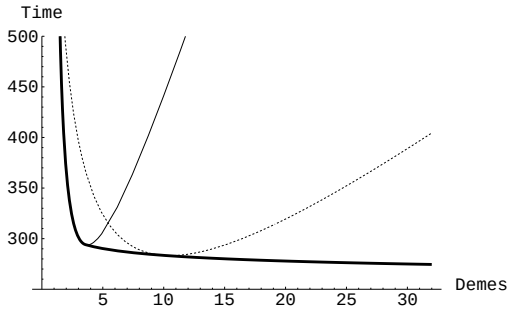


Figure 5: The execution time using the optimal degree decreases very slowly (bold line). It is bounded (and well approximated) by the optimal time of the fully connected topology (thin line). See the text for an explanation of the points at $r = 2, 3$. The execution time of a topology of degree $r/4$ (dotted line) is plotted as an example.

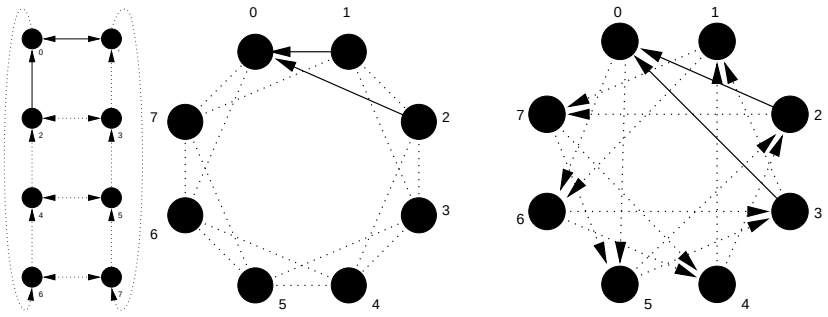
In this example, $\delta^* \approx 2.5$, which should be interpreted as $\delta^* = 3$. However, it is not possible to connect $r = 2$ or $r = 3$ demes together using more than $r - 1$ edges, and therefore the first two points in the plot of the optimal time correspond to a fully connected topology.

Although the fully connected topology cannot integrate many demes efficiently, its *optimal* configuration is a good choice to reduce the execution time. Optimally configured topologies that use more demes reach solutions faster, but the reductions are not substantial. In fact, since the optimal time decreases very slowly when more than r^* demes are used, after that point, the efficiency drops almost linearly with the number of demes. However, we shall see that optimally configured topologies can reduce the execution time significantly after multiple epochs.

6 Considering Multiple Epochs

The previous section showed how to find the optimal degree of connectivity that minimizes the execution time for a particular domain. However, even with a fixed degree there are $\binom{r-1}{\delta}$ ways to connect the demes, and we still face the question of how to choose a particular topology. Certainly, if the algorithm is only executed for two epochs, it does not matter how the demes are connected, because only the immediate neighbors affect the search. After more than two epochs, however, a deme receives indirect contributions from other demes. The purpose of this section is to quantify the effect of those contributions on the quality of the search, and to determine how to minimize the execution time after multiple epochs. Some of these results can also be found in Cantú-Paz (1999c).

Consider the topologies with degree $\delta = 2$ depicted in Figure 6. These are only three of the $\binom{7}{2} = 21$ possible topologies of degree two. Figure 7 shows the results of experiments with a 20-BB 4-bit trap function on eight demes connected with the +1+2 and +2+3 topologies of figure 6 and a bi-directional ring. The results are averaged over 100 repetitions at each deme size; the demes used pairwise tournament selection, two-point crossover with probability 1.0, and no mutation. The migration rate was set to its maximal value of 1/3. Figure 7 shows the proportion of correct BBs per deme after one, two, three, and four epochs. The quality of the solutions improves after successive epochs, and the largest increase occurs after the second epoch. As one would expect, the results for different



(a) Ladder. (b) +1+2 topology. (c) +2+3 topology.

Figure 6: Different topologies with two neighbors.

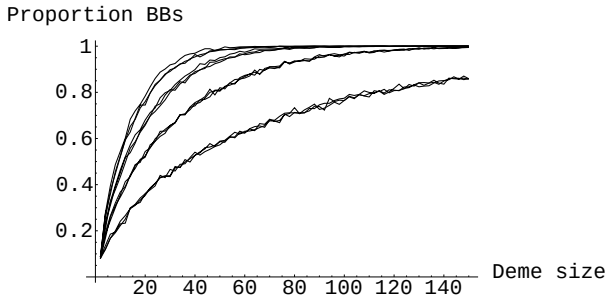
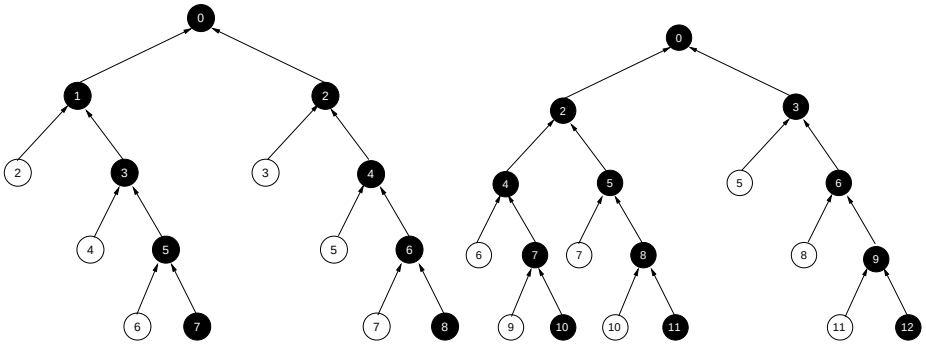


Figure 7: Average quality per deme after one, two, three, and four epochs (from right to left) using eight demes connected with different topologies of degree two.



(a) +1+2 Topology. (b) +2+3 Topology.

Figure 8: Tree representations of the extended neighborhood of deme 0 of two different topologies of degree two with 16 demes. The black nodes represent the new members of the extended neighborhood after each epoch. The white nodes represent demes that already belong to the extended neighborhood, and they are not expanded to avoid clutter in the graphs.

topologies after the first two epochs are indistinguishable, and the difference after three and four epochs is very small. This observation will be used to derive a model of solution quality that depends only on the degree of the connectivity graph and on the migration rate, but that ignores the specific topology. Before doing so, we first introduce the concept of the extended neighborhood of a deme.

6.1 Extended Neighborhoods

To visualize how the choice of topology affects the quality of the search, imagine a tree rooted on a particular deme. The descendants of a node in the tree are the immediate neighbors of the deme it represents, and the τ-th level in the tree contains the demes that are reachable from the root deme after τ epochs. These demes form the *extended* neighborhood of the root and are taken into account only the first time they are reached.[1] Figure 8 shows two such trees that correspond to the +1+2 and +2+3 topologies with 16 demes.

A simple way to bound the contribution from the extended neighborhood is to assume that the demes form panmictic groups as soon as they come in contact with others. In this view, after τ epochs, the aggregate population size would be $r_\tau n_d$, where r_τ is the number of demes in the extended neighborhood after the τ-th epoch, and n_d is the size of each deme. Under this assumption, the solution quality would be given by $P_{bb}(r_\tau n_d)$. Of course, demes do not become panmictic as soon as they reach one another, and therefore the size of the extended neighborhood should be adjusted with a mixing coefficient $c_m < 1$. Then,

[1] More formally, the definition of the extended neighborhood is as follows. Consider a directed graph $G = (V, E)$, where V is the set of vertices that represent the demes, and E is the set of edges that represent connections between demes. The extended neighborhood of a deme v is the set $R_\tau = \bigcup_{i=0}^{\tau} a : a \xrightarrow{i} v$, where $a \xrightarrow{i} v$ denotes a path of length i from a to v. $r_\tau = |R_\tau|$.

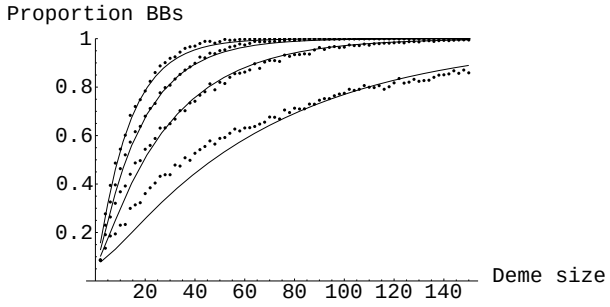


Figure 9: Theoretical predictions (line) and experimental results (dots) of the average quality per deme after 1, 2, 3, and 4 epochs (from right to left) using eight demes connected by a +1+2 topology.

the quality becomes $P_{bb}(c_m r_\tau n_d)$.

6.2 Designing for Multiple Epochs

The observation that topologies of the same degree reach almost identical solutions has an important implication: if an accurate quality predictor can be derived for one topology, it would be accurate for any topology of the same degree.

Some topologies are easier to study than others, because the size of their extended neighborhoods increases in a regular form. In particular, there are topologies where the size of the extended neighborhood is simply $r_\tau = \delta(\tau - 1) + 1 = \delta\tau' + 1$. Examples of such topologies are the +1+2 binary topology depicted in Figure 6 and a +1+2+3 ternary topology. We will use this class of topologies to study the effect of the degree of the network on the solution quality after several epochs.

Using the simple model introduced in the previous subsection, the quality after τ epochs is $P_{bb_\tau} = P_{bb}(c_m r_\tau n_d)$. For simplicity, we use $n_\tau = c_m r_\tau n_d$ to represent the number of individuals in the extended neighborhood. The key to obtain an accurate quality prediction is to adjust n_τ with an appropriate c_m. We can deduce the value of the mixing coefficient by considering some of the properties it should have. For instance, n_τ should increase as τ grows, and when $\tau = 1$, the value of n_τ should be equal to n_d, because the demes are isolated during the first epoch. In addition, the previous section showed that the deme size $n_d \propto \frac{1}{\sqrt{\delta}}$ (Equation 13). Putting everything together, we may write n_τ as:

$$n_\tau = (\sqrt{\delta}\tau' + 1)n_d, \tag{16}$$

which means that $c_m = \frac{\sqrt{\delta}\tau'+1}{\delta\tau'+1} \approx \frac{1}{\sqrt{\delta}}$. Experimental tests were performed to assess the accuracy of this model. The experiments use eight demes connected by a +1+2 topology and the same test function and experimental conditions as previous experiments in this paper: 100 repetitions at each deme size, pairwise tournament selection, two-point crossover with probability 1.0, and no mutation. The quality was measured at the end of the first four epochs. Figure 9 shows that the predictions of $P_{bb}(n_\tau)$ match very well the experimental results.

The next step is to find a deme-sizing equation. The procedure is straightforward using the gambler's ruin model. Making $P_{bb}(n_\tau) = 1 - (\frac{q}{p})^{n_\tau/2^k} = \hat{P}$ and solving for n_d results in

$$n_d = \frac{1}{\sqrt{\delta\tau'} + 1} \frac{2^k \ln(1 - \hat{P})}{\ln \frac{q}{p}}, \tag{17}$$

which can be rewritten in a more compact form by grouping all the problem-dependent constants (the second term above) into one constant n_0, so $n_d = \frac{n_0}{\sqrt{\delta\tau'}+1}$.

This form of the deme size with $\tau = 2$ is similar to the equation found in the previous section. With a closed-form expression for the deme size after multiple epochs, the execution time of the parallel GA may be easily minimized. In this case, the time is

$$T_p = \tau \left(g n_d T_f + \delta T_c \right), \tag{18}$$

and τ is restricted by Equation 22. To simplify the calculations, n_d may be approximated as $\frac{n_0}{\sqrt{\delta\tau'}}$. Making $\frac{\partial T_p}{\partial \delta} = 0$ and solving for δ gives the optimal degree of the topology as

$$\delta^* = \left(\frac{g n_0 T_f}{2\tau' T_c} \right)^{2/3}, \tag{19}$$

which is equivalent to the optimum found in the previous section when $\tau = 2$.

Note that δ^* depends very weakly on r. The dependency on r is in the $\ln(1 - \hat{P})$ term of n_0. $\hat{P}$ is approximately $\frac{Q}{m} - \frac{\sqrt{\ln r}}{\sqrt{2m}}$, so the dependency of the optimal degree on r is $O(\ln \ln r)$. This is significant, because if δ^* does not change much as more demes are used, then the execution time (Equation 18) would not change much either. However, we should not to dismiss the algorithm's capability to reduce the execution time. After all, δ^* depends strongly on the number of epochs τ, and both δ^* and the execution time will decrease significantly as more epochs are used.

This raises another point: sometimes the choice of topology is restricted by hardware constraints. In this case, δ may be considered to be constant and the execution time may be optimized with respect to τ. Making $\frac{\partial T_p}{\partial \tau} = 0$ and solving for τ gives the optimal number of epochs:

$$\tau^* = 1 + \sqrt{\frac{g n_0 T_f}{\delta^{3/2} T_c}} \tag{20}$$

The corresponding deme size may be found by substituting δ^* (or τ^*) in Equation 17.

7 Parallel Demes in the Long Run

This section focuses on an important property of parallel GAs with multiple populations. In the long run, r demes of size n_d that repeatedly communicate with each other after

convergence with the maximum migration rate possible, reach the same solution as a simple GA with a population with rn_d individuals (Cantú-Paz, 1998a, 1998b). We denote τ_c as the critical number of epochs necessary to reach a solution as good as a GA with an aggregate population.

Recall that the execution time of the parallel GA depends directly on the number of epochs (Equation 18). When τ_c epochs are used, the size of the demes is reduced to $n_d = \frac{n}{r}$, so the execution time becomes:

$$T_p = \tau_c \left(\frac{gnT_f}{r} + \delta T_c \right) \tag{21}$$

Our objective is to minimize this time. All of the terms of this equation are known, except for τ_c. To derive an estimate of τ_c, note that $\lim_{\tau \to \infty} P_{bb}(n_\tau) = 1$, because n_τ grows without limit as more epochs are used, which is incorrect. The derivation of n_τ is based on the concept of extended neighborhoods, but physically, the size of the extended neighborhood is bounded by the sum of the sizes of all the demes (rn_d). Making $n_\tau = rn_d$ and solving for τ gives an approximation of the number of epochs it takes the algorithm to converge:

$$\tau_{max} = \frac{r-1}{\sqrt{\delta}} + 1. \tag{22}$$

Substituting $\tau_c = \tau_{max}$ into Equation 21, making $\frac{\partial T_p}{\partial r} = 0$, and solving for r results in the optimal number of demes:

$$r^* = \sqrt{g \frac{\sqrt{\delta} - 1}{\delta}} \sqrt{\frac{nT_f}{T_c}} \tag{23}$$

Note that the second term also appears in the optimal number of processors of single-population parallel GAs, suggesting that, asymptotically, the two types of parallel GAs can use the same number of processors to reduce the execution time. However, the question of which algorithm is the fastest depends on the particular application. Users can use the equations presented here to estimate the execution time of the two types of parallel GAs and decide which one to use.

8 Summary and Conclusions

The paper began with an examination of the execution time of parallel GAs with a single population. The calculations focused on an implementation where the population is distributed among several processors, but the general form of the time is also applicable to more conventional master-slave implementation. The main result is that the optimal number of processors is $O(\sqrt{\frac{nT_f}{T_c}})$. Since n and T_f increase as the problem becomes more difficult, this result suggests that single-population parallel GAs can use a large number of processors to reduce the execution time.

The single distributed population closely resembles multiple populations that communicate every generation using a fully connected topology and the maximal migration rate. Therefore, the previous calculation of the optimal number of processors also serves as an indication of the scalability of multi-deme parallel GAs.

The rest of the paper dealt with parallel GAs with multiple populations and arbitrary topologies, but migration occurs only after the demes converge, which is a lower bound of the migration frequency. The gambler's ruin model was used to derive an accurate prediction of the solution quality after multiple epochs. The models should be directly applicable to any additively decomposable function. The quality model was first used to determine the optimal topology when the number of demes is fixed, and later to calculate the optimal number of processors when the topology is fixed and the algorithm is executed until all the populations converge to the same solution. The result is that the optimal number of populations is also $O(\sqrt{\frac{nT_f}{T_c}})$.

The results of this study suggest that, regardless of their type, parallel GAs can integrate large numbers of processors and reduce significantly the execution time of many practical applications. Although parallel GAs have long been regarded as highly scalable, the closed-form expressions of the optimal number of processors make it possible to quantify those claims taking into account the particular problem domain and the hardware platform.

Acknowledgments

Erick Cantú-Paz was partially supported by a Fulbright-García Robles Fellowship. The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0050. Research funding for this project was also provided by a grant from the US Army Research Laboratory Program, Cooperative Agreement DAAL01-96-2-003. The US 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 and endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the US Government.

References

Bethke, A. D. (1976). *Comparison of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity*. Technical Report 197, Logic of Computers Group, University of Michigan, Ann Arbor, Michigan.

Beyer, H.-G. (1993). Toward a theory of evolution strategies: Some asymptotical results from the (1+λ)-Theory. *Evolutionary computation*, 1(2):165–188.

Braud, A. and Vrain, C. (1999). A parallel genetic algorithm based on the BSP model. In *Evolutionary Computation and Parallel Processing Workshop*, *Proceedings of the 1999 GECCO Workshops*, Morgan Kaufmann, San Mateo, California.

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

Cantú-Paz, E. (1998a). *A Markov chain analysis of parallel genetic algorithms with arbitrary topologies and migration rates*. IlliGAL Technical Report 98010, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Cantú-Paz, E. (1998b). Using Markov chains to analyze a bounding case of parallel genetic algorithms. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H. and Riolo, R. L., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 456–462, Morgan Kaufmann, San Francisco, California.

Cantú-Paz, E. (1999a). Migration Policies and Takeover Times in Parallel Genetic Algorithms. In Banzhaf, W., editor, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, page 775, Morgan Kaufmann, San Mateo, California.

Cantú-Paz, E. (1999b). *Migration policies, selection pressure, and parallel evolutionary algorithms*. IlliGAL Technical Report 99015, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Cantú-Paz, E. (1999c). Topologies, migration rates, and multi-population parallel genetic algorithms. In Banzhaf, W., editor, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 91–98, Morgan Kaufmann, San Mateo, California.

Cantú-Paz, E. and 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. and Riolo, R., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 353–361, Morgan Kaufmann, San Francisco, California.

Cantú-Paz, E. and Goldberg, D. E. (1997b). Predicting speedups of idealized bounding cases of parallel genetic algorithms. In Bäck, T., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 113–121, Morgan Kaufmann, San Francisco, California.

Cantú-Paz, E. and Goldberg, D. E. (in press). Parallel genetic algorithms: theory and practice. *Computer Methods in Applied Mechanics and Engineering*.

Feller, W. (1966). *An introduction to probability theory and its applications*. 2d ed. Volume 1. John Wiley and Sons, New York, New York.

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

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

Grefenstette, J. J. (1981). *Parallel adaptive algorithms for function optimization*. Technical Report CS-81-19, Computer Science Department, Vanderbilt University, Nashville, Tennessee.

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

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

Harik, G., Cantú-Paz, E., Goldberg, D. E. and Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pages 7–12, IEEE Press, Piscataway, New Jersey.

Harter, H. L. (1970). *Order statistics and their use in testing and estimation*, U.S. Government Printing Office, Washington, D.C.

Lin, S.-C., Goodman, E. D. and Punch III, W. F. (1997). A genetic algorithm approach to dynamic job shop scheduling problems. In Bäck, T., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 481–488), Morgan Kaufmann, San Francisco, California.

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

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

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

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

Valenzuela-Rendón, M. (1989). *Two analysis tools to describe the operation of classifier systems*. Doctoral dissertation, University of Alabama, Tuscaloosa, Alabama. Also available as TCGA Report 89005.