Efficient Parallel Genetic Algorithms: Theory and Practice

Erick Cantú-Paz

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

David E. Goldberg

Department of General Engineering and
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
deg@illigal.ge.uiuc.edu

Abstract

Parallel genetic algorithms (GAs) are complex programs that are controlled by many parameters, which affect their search quality and their efficiency. The goal of this paper is to provide guidelines to choose those parameters rationally. The investigation centers on the sizing of populations, because previous studies show that there is a crucial relation between solution quality and population size. As a first step, the paper shows how to size a simple GA to reach a solution of a desired quality. The simple GA is then parallelized, and its execution time is optimized. The rest of the paper deals with parallel GAs with multiple populations. Two bounding cases of the migration rate and topology are analyzed, and the case that yields good speedups is optimized. Later, the models are specialized to consider sparse topologies and migration rates that are more likely to be used by practitioners. The paper also presents the additional advantages of combining multi- and single-population parallel GAs. The results of this work are simple models that practitioners may use to design efficient and competent parallel GAs.

1 Introduction

To solve some problems, genetic algorithms (GAs) may require hundreds or thousands of function evaluations. Depending on the cost of each evaluation,

it may take the GA days, months, or years to find an acceptable solution. Fortunately, GAs work with a population of independent solutions, which makes feasible to distribute the computational load among several processors. Indeed, some may say that GAs are "embarrassingly parallel" programs, and that it is trivial to make fast parallel GAs. But, although the mechanics of parallel GAs are simple, they are complex non-linear algorithms that are controlled by many parameters that affect the quality of their search and their efficiency.

In particular, the design of parallel GAs involves choices such as using one population or multiple populations. In both cases, the size of the population(s) must be determined carefully, and when multiple populations are chosen, one must also decide how many to use. Also, the populations may remain isolated or they may communicate. Communication involves extra costs and additional decisions on topologies, on how many individuals are exchanged, and on the frequency of communications.

Traditionally, these parameters are set by experimentation or are found just by chance. But these approaches may result in an inefficient use of computing resources, in an inadequate search quality, or both. Therefore, practitioners may regard parallel GAs as being impractical, or may find that they do not work satisfactorily. To make parallel GAs efficient, accurate, and reliable we must go beyond this ad hoc tuning of parameters, and, to that effect, this paper presents a simple and confirmed theory that users can use to configure and size their programs.

Our research focuses on sizing the populations correctly, because previous studies show that there is a fundamental relation between the population size and the solution quality. The paper is organized as follows. Section 2 shows how to size a simple GA to reach a solution of a certain quality, and then section 3 analyzes a parallel implementation of the simple GA. The rest of the paper considers parallel GAs with multiple populations. Section 4 presents population sizing equations and calculates the expected speedups for two idealized bounding cases. It also describes how to optimize the bounding case with the best speedup. Section 5 shows that combining single- and multiple-population parallel GAs results in a faster hierarchical algorithm. The last part of the paper specializes the models for multi-population GAs to consider topologies and migration rates that practitioners are likely to use. We finish this report with a summary and the conclusions of our study.

2 Simple Genetic Algorithms

The population of a genetic algorithm is formed by individuals with artificial chromosomes that encode solutions of the problem that we want to solve. Each

individual is evaluated to determine how well it solves the problem, and the best solutions are selected to recombine with others. The basic mechanism in GAs is that of Darwinian evolution: good traits survive and mix to form new—and possibly better—individuals, while selection eliminates the bad traits from the population.

The notion of 'good' traits is formalized with the concept of building blocks (BBs), which are templates (formally known as schemata) that specify a well-adapted set of features common to good solutions. The prevailing theory suggests that GAs work by identifying and recombining BBs using selection and crossover. In our discussion we restrict the notion of BB to the lowest-order schemata that must be identified in order to reach the global optimum [21].

One of the most important decisions needed to use a GA is to set the number of individuals in the population. If the population is too small, there might not be an adequate supply of BBs, and it will be difficult to identify good solutions. On the other hand, if the population is too big the GA will waste time processing unnecessary individuals, and this may result in unacceptably slow performance. The same problems are encountered in parallel GAs, and therefore the results of the next section are important to our quest for efficient parallel GAs.

2.1 The Gambler's Ruin Model

To predict the quality of the solution of a simple GA, Harik, Cantú-Paz, Goldberg, and Miller [17] modeled the selection mechanism of GAs as a one-dimensional random walk. The 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. It focuses on only one partition of order k, and it assumes that decisions are independent across partitions. The key idea is to consider the fitness contributions from the other partitions as noise that interferes in the decision process.

The number of copies of the correct BB is represented by the position, x, of a particle on a one-dimensional space, as depicted in figure 1. Absorbing barriers at $x = 0$ and $x = n$ bound the space and 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 BBs in a randomly initialized population, which in a domain using binary encoding is $x_0 = \frac{n}{2^k}$.

At each step of the random walk there is a chance of obtaining or loosing one copy of the right BB. The selection mechanism is supposed to choose the individuals that have the best BBs and to eliminate the others, but sometimes the wrong individuals are chosen. To understand why this may occur consider

Fig. 1. The bounded one-dimensional space of the gambler's ruin problem.

a competition between an individual i_1 that contains the optimal BB in a partition, H_1, and an individual i_2 with the second best BB, H_2. The probability of deciding correctly between these two individuals is the probability that the fitness of i_1 (f_1) is greater than the fitness of i_2 (f_2), or equivalently the probability that $f_1 - f_2 > 0$.

Assuming that the fitness is an additive function of the contributions of independent subfunctions, we may consider that the fitness distributions of H_1 and H_2 are normal by the central limit theorem. Since the fitness distributions are normal, the distribution of $f_1 - f_2$ is itself normal and has known properties: the mean is the difference of the individual means, and the variance is the sum of the individual variances. Therefore,

$$f_1 - f_2 \sim N(\overline{f_{H_1}} - \overline{f_{H_2}}, \sigma^2_{H_1} + \sigma^2_{H_2}).$$

Substituting $d = \overline{f_{H_1}} - \overline{f_{H_2}}$ in the expression above and normalizing, the probability of making the correct decision on a single trial is

$$p = \Phi\left(\frac{d}{\sqrt{\sigma^2_{H_1} + \sigma^2_{H_2}}}\right), \tag{1}$$

where Φ is the cumulative distribution function for a normal distribution with zero mean and a standard deviation of one.

In domains where the m partitions are uniformly scaled, the total noise coming from the $m' = m - 1$ partitions that are not competing directly is $\sigma^2_H = m'\sigma^2_{bb}$, where σ^2_{bb} is the average variance of a partition. Therefore, the probability of making the right choice in a single trial becomes

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

A well-known result about random walks is the probability that a particle will eventually be captured by the absorbing barrier at $x = n$ [11]:

$$P_{bb} = \frac{1 - \left(\frac{q}{p}\right)^{x_0}}{1 - \left(\frac{q}{p}\right)^{n}}, \tag{3}$$

where p is the probability of gaining a copy of the BB in a particular competition (given by equation 2), and $q = 1 - p$. From this equation it is relatively easy to find an expression for the population size. First, note that for increasing values of n, the denominator in equation 3 approaches 1 very quickly and it can be ignored in the calculations. Since the BBs are independent of each other, the expected number of partitions with the correct BB at the end of a run is $E(BBs) = mP_{bb}$. Assuming that we are interested in finding a solution with $\hat{Q}$ partitions correct, we can solve $P_{bb} = \frac{\hat{Q}}{m}$ for n and obtain a population sizing equation:

$$n = \frac{2^k \ln(\alpha)}{\ln\left(\frac{1-p}{p}\right)}, \tag{4}$$

where $\alpha = 1 - \frac{\hat{Q}}{m}$. To observe more clearly the relations between the population size and all the variables involved, we can expand p and write the last equation in terms of the signal, the noise, and the number of partitions in the problem. First, approximate p using the first two terms of the power series expansion for the normal distribution as $p = \frac{1}{2} + \frac{1}{2}x$ [1], where $x = d/(\sigma_{bb}\sqrt{\pi m'})$. Substituting this approximation for p into equation 4 results in

$$n = 2^k \ln(\alpha) / \ln\left(\frac{1-x}{1+x}\right). \tag{5}$$

Since x tends to be a small number, $\ln(1-x)$ may be approximated as $-x$ and $\ln(1+x)$ as x. Using these approximations and substituting x into the equation above gives

$$n = -2^{k-1} \ln(\alpha) \frac{\sigma_{bb}\sqrt{\pi m'}}{d}. \tag{6}$$

This equation quantifies many intuitive notions that practitioners have about problem difficulty for GAs. For example, problems with longer BBs (higher k) are more difficult to solve than problems with short BBs, because long BBs are scarce in a randomly initialized population. The equation also shows how the population size increases when the signal-to-noise ratio is low. Intuitively, problems with a high variability are harder because it is difficult for

the algorithm to detect the signal from good solutions when the interference from not-so-good solutions is high. Similarly, problems with many partitions are more difficult than problems with a few partitions because there are more sources of noise.

2.2 Experimental Verification

This section shows that the gambler's ruin model can predict accurately the quality of the solution reached by a simple GA. We present experimental results with functions of varying difficulty that are also used to test the parallel GA models presented below. The population sizes required to solve the problems vary from a few tens to a few thousands, demonstrating that the predictions of the model scale well with increasing problem difficulty.

All empirical results in this paper are averaged over 100 independent runs. The GAs use pairwise tournament selection and no mutation. Each run was terminated when the population had converged completely (which is possible because the mutation rate is zero), and we report the percentage of partitions that converge to the correct value.

The two sets of experiments use deceptive trap functions. Fully deceptive trap functions are used in many studies of GAs because their difficulty is well understood and they can be regulated easily [10]. The first deceptive test function is based on the 4-bit function depicted in figure 2. The value of this function depends on the number of bits set to one; the fitness increases with more bits set to zero until it reaches a local (deceptive) optimum. The global maximum of the function occurs precisely at the opposite extreme where all four bits are set to one, so an algorithm cannot use any partial information to find it. Therefore, the shortest BBs that need to be identified to reach the global optimum are of order $k = 4$. The signal difference d (the difference between the optimal and the deceptive maxima) is 1, and the fitness variance (σ^2_{bb}) is 1.215. Our test function is formed by concatenating $m = 20$ copies of the trap function for a total string length of 80 bits. The second deceptive test function is formed by concatenating 10 copies of an 8-bit trap. In this case, the signal difference is the same as before ($d = 1$), but the variance is higher ($\sigma^2_{bb} = 2.1804$).

The experiments with the 4-bit trap function use two-point crossover with probability 1.0, and there is no mutation. Since the 8-bit function has longer BBs, we use 1-point crossover to avoid excessive disruption of the BBs. Figure 3 presents the prediction of the percentage of BBs correct at the end of the run along with the results from the experiments. The bold lines are the prediction of the random walk model and the thin lines are the prediction of a previous

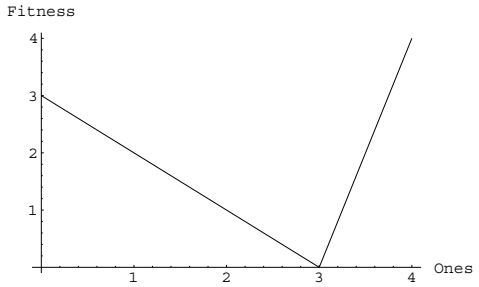
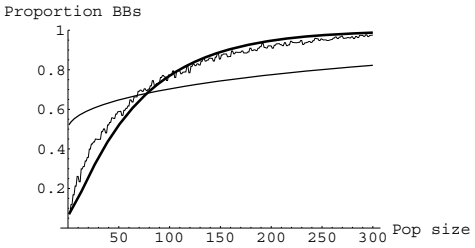
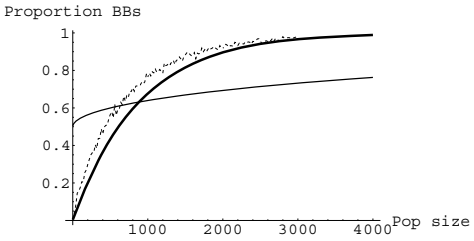


Fig. 2. A 4-bit fully deceptive function.



(a) 4-bit trap function with $m = 20$ BBs.



(b) 8-bit trap function with $m = 10$ BBs.

Fig. 3. Results comparing theory and experimental results for deceptive trap functions. Equation 3 (in bold) predicts well the experimental results (dotted). The thin line is the previous decision-making model of Goldberg, Deb, and Clark.

population sizing model by Goldberg, Deb, and Clark [14].

A correctly sized population is the first step toward efficient genetic algorithms. The next section investigates how to make single-population GAs faster by parallelizing the evaluation of fitness.

3 Master-Slave Parallel GAs

Probably the easiest way to parallelize GAs is to distribute the evaluation of fitness among several slave processors, while one master processor executes the GA operations (selection, recombination, and mutation). Master-slave GAs have many advantages: they explore the search space in exactly the same manner as a serial GA, they are very easy to implement, and significant improvements in performance are possible in many cases. This section examines the execution time of a simple master-slave implementation. Even though we can conceive faster implementations, the objective of this section is to give a lower bound on the performance improvements that are possible by any parallel GA.

There are many examples of master-slave GAs in the literature. Early studies by Bethke [3] and by Grefenstette [15] speculated about the possible benefits that even simple parallel implementations would give to domains with long evaluation times. But despite the early promises of great speedups, several researchers who used master-slave GAs stumbled upon a scalability problem: as more processors were used the efficiency of the parallel algorithm diminished. In most cases the cause for the diminishing efficiency was the increase in communication costs (e.g.,[12], [19], and [2]).

We present an analysis that examines one generation of a master-slave GA, and shows how to minimize its execution time. The calculations consider the cost of communications, but ignore the time used by selection, crossover, and mutation, because we assume they are much smaller than the times used to evaluate and communicate individuals. The analysis also assumes that the number of individuals assigned to any processor is constant, and that the evaluation time is the same for each individual.

3.1 Analysis

The analysis of the execution time centers on the master processor. Figure 4 depicts the sequence of events during one generation. First, the master sends a fraction of the population to each of the $\mathcal{S}$ slaves, using time T_c to communicate with each. Next, the master evaluates a fraction of the population using time $\frac{nT_f}{\mathcal{P}}$, where T_f is the time required to evaluate one individual, n is the size of the population, and $\mathcal{P} = \mathcal{S} + 1$ is the number of processors used. The slaves start to evaluate their portion of the population as soon as they receive it, and return the evaluations to the master as soon as they finish. The last slave and the master finish their evaluations at the same time, but there is a delay of time T_c before the master receives the evaluations and can proceed. Considering all the contributions from communications and computations, the total elapsed time for one generation is

$$T_p = \mathcal{P}T_c + \frac{nT_f}{\mathcal{P}}. \tag{7}$$

As more slaves are used, the computation time decreases as desired, but the communications time increases. This tradeoff entails the existence of an optimal number of processors that minimizes the execution time. To find the optimal make $\frac{\partial T_p}{\partial \mathcal{P}} = 0$ and solve for $\mathcal{P}$:

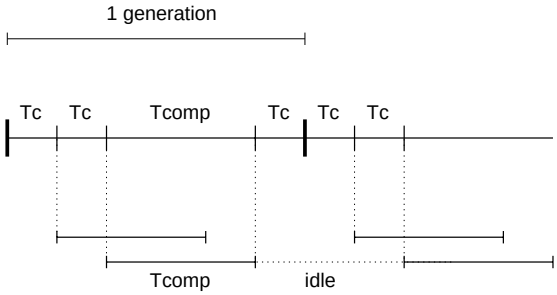


Fig. 4. A schematic of the execution of a master-slave parallel GA when the master evaluates a fraction of the population.

$$\mathcal{P}^* = \sqrt{\frac{nT_f}{T_c}}, \tag{8}$$

which can be expressed in a more compact form as $\mathcal{P}^* = \sqrt{\gamma n}$, where $\gamma = T_f/T_c$. The optimal number of slaves is $\mathcal{S}^* = \mathcal{P}^* - 1$.

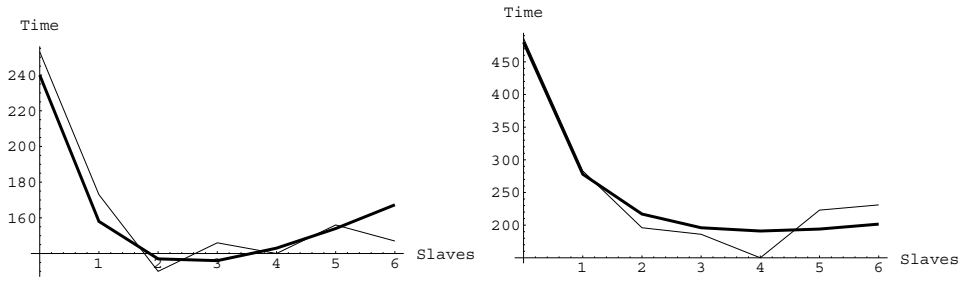
An important concern on implementing master-slave parallel GAs is that the frequent communication may offset any gains in computation time. The time that a simple GA uses in one generation is $T_s = nT_f$, and to ensure that the parallel implementation has a better performance than a simple GA the following relationship must hold:

$$Sp = \frac{T_s}{T_p} = \frac{nT_f}{\frac{nT_f}{\mathcal{P}} + \mathcal{P}T_c} = \frac{n\gamma}{\frac{n\gamma}{\mathcal{P}} + \mathcal{P}} > 1. \tag{9}$$

This ratio is the parallel speedup for the master-slave parallel GA, and it formalizes the intuitive notion that master-slave GAs do not benefit problems with very short evaluation times. Another concern when implementing parallel algorithms is to keep the processor utilization high. Formally, the efficiency of a parallel program is defined as the ratio of the parallel speedup over the number of processors:

$$E_f = \frac{T_s}{T_p\mathcal{P}}. \tag{10}$$

Ideally, the parallel speedup should be equal to the number of processors used, and the efficiency should be one. In reality, the cost of communications causes the efficiency to decrease as more processors are used. To find the critical number of slaves that maintain a predetermined efficiency $\widehat{E_f}$ make equation 10 equal to $\widehat{E_f}$ and solve for $\mathcal{P}$:



(a) $T_f = 2ms$ (b) $T_f = 4ms$

Fig. 5. Elapsed time (ms) per generation of a master-slave GA. The thick lines are the theoretical predictions and the thin lines are the experimental results.

$$\mathcal{P}_c = \sqrt{\frac{1 - \widehat{E_f}}{\widehat{E_f}}} \sqrt{\frac{nT_f}{T_c}},$$

which is a fraction of the optimal number of processors $\mathcal{P}^*$. Note that $\mathcal{P}_c = \mathcal{P}^*$ when the efficiency is 0.5, so the maximum speedup achievable by a master-slave parallel GA can be easily computed as $Sp^* = 0.5\mathcal{P}^*$.

The simple calculations in this section represent a lower bound on the potential speedups of parallel GAs. Better master-slave implementations and other parallel GAs should be able to use more processors to their advantage, and should do no worse than the simple case examined here.

3.2 Experiments

This section describes experiments with a master-slave implementation on a network of IBM RS6000 workstations. The workstations are connected by a 10 Mbits/sec Ethernet and all communications are implemented using PVM 3.3. We use the same setup for all the parallel experiments in the paper. This is a rather slow communications environment, and no performance improvements are expected when simple test functions are used. For this reason, the experiments use an artificial function that can be altered easily to change its evaluation time (T_f).

The artificial test problem is a dummy function that consists in a loop with a single addition that can be repeated an arbitrary number of times. The length of the individuals was set to 80 bytes and the population size to 120 individuals. The master-slave GA was executed for 10 generations and the results reported are the average of 30 runs. We determined empirically that the cost of communications in our system was approximately $T_c = 19$ milliseconds.

For the first experiment the evaluation time of the test function was set to 2 milliseconds. The optimal number of slaves for this problem may be calculated as as $\mathcal{S}^* = \sqrt{\frac{nT_f}{T_c}} - 1 = \sqrt{\frac{120(2)}{19}} - 1 = 2.55$. In the second experiment we double the evaluation time of the test function to 4 milliseconds, and in this case the model predicts that the optimal number of slaves is $\mathcal{S}^* = 4.02$. Figure 5 shows the elapsed time per generation of the master-slave parallel GA along with the theoretical predictions (using equation 7).

These results show that the theory predicts the execution time and the optimum number of slaves quite accurately. Now we have a baseline against which we can compare the performance of parallel GAs, and we proceed to examine parallel GAs with multiple populations.

4 Bounding Cases of Multiple-Deme GAs

Multiple-deme parallel GAs are also called "coarse-grained" or "distributed" GAs, because the communication to computation ratio is low, and they are often implemented on distributed-memory computers. They are also known as "island model" GAs, because they resemble a model that is used to describe natural populations isolated by the distance between them (as in islands). Both in the model and in the GAs individuals may migrate occasionally to any population.

Designing multiple-deme parallel GAs that reach good solutions fast is hard because it involves many difficult and interrelated choices. The three main problems are to determine (1) the size and the number of demes, (2) the topology of the connections between the demes, and (3) how many individuals migrate each time.

These three dimensions of the problem form a natural decomposition which is used in the remainder of the paper. We recognize that the issues are not independent, but in our work we try to isolate them as much as possible to construct simple models that tell us something about the effect of the parameters on quality and efficiency.

Most studies of parallel GAs are empirical investigations that concentrate on the choices of topology and migration rates and treat the population sizing issue as a secondary problem. But, since the population size largely determines the quality of the solution and the duration of the run, it seems natural to us to emphasize the correct sizing of populations in our research.

However, we do not ignore the importance of topologies and migration rates, and to make progress in the study of deme sizes we consider two idealized

bounding cases. The first bound is a set of simple GAs running in parallel with no interactions between them. In the second bounding case, each deme exchanges individuals with all the others, and the migration rate is set to a maximum value. Although it is likely that users set the connectivity and migration to intermediate values, the bounding cases serve as indicators of the performance of parallel GAs. Later, the models will be specialized to consider sparser topologies and lower migration rates.

Despite the difficulties in their design, multiple-deme parallel GAs have been very popular, and the literature contains many reports of successful implementations. Here we only mention a few studies that are closely related to our work. Further examples can be found in a review by Cantú-Paz [6].

Grosso experimented with isolated demes and with a "delayed" migration scheme in which communications began only after the demes were near convergence [16]. At that point, the demes exchanged individuals with a high migration rate. Grosso found that the solution found by isolated demes was much lower than that reached with a single large population. However, his delayed scheme found solutions of the same quality as the panmictic population and as multiple demes with frequent migrations. Braun presented an algorithm where migration occurred after the demes converged completely [5], and later Munetomo, Takai, and Sato used a similar strategy [20].

4.1 Isolated Demes

The first bounding case of parallel GAs considers that the demes evolve in complete isolation. Without communication, the migration rate is zero, and this is clearly a lower bound. Also, no connections between the demes represent a lower bound in the connectivity of the topology. Cantú-Paz and Goldberg [8] performed an analysis that is equivalent to the one presented in this section.

The first step in our analysis is to determine the target quality $\hat{P}$ required in each deme. We may be conservative and use the required solution's quality $\hat{Q}$, but in a run with multiple demes the chance that at least one of them succeeds increases as more demes are used. Therefore, the deme's target quality can be relaxed, and the following paragraphs show how to compute that relaxation.

The quality of the solution is measured as the number of partitions (X) that converge correctly, and under the assumption that partitions are independent from each other, the quality has a binomial distribution. We may write the qualities of the solutions of the r demes in ascending order as

$$X_1, X_2, ..., X_r.$$

Table 1
Expected values of the highest order statistic $\mu_{r:r}$ of a normal distribution with zero mean and unit standard deviation for representative values of r.

r	1	2	4	8	16	32	64	128	256
$\mu_{r:r}$	0	0.564	1.029	1.423	1.766	2.069	2.343	2.594	2.826

These are the order statistics of the quality of the solution, and we are interested in designing the parallel GA so that the expected value of X_r (the best solution found by the r demes) be equal to $\hat{Q}$. Unfortunately, there is no closed-form expression for the mean values of the maximal order statistics of samples greater than five, but these values have been tabulated extensively for the standard normal distribution (see table 1, taken from [18]). To take advantage of this, the binomial distribution of the quality may be approximated with a Gaussian distribution, and the number of partitions correct is normalized as $z_i = \frac{X_i - mP_{bb}}{\sqrt{mP_{bb}(1-P_{bb})}}$. The expected quality of the best deme is $E(z_r) = \mu_{r:r}$, where $\mu_{r:r}$ denotes the mean of the highest-order statistic of a standard normal distribution. The expected value of X_r is

$$\hat{Q} = E(X_r) = mP_{bb} + \mu_{r:r}\sqrt{mP_{bb}(1-P_{bb})}. \tag{11}$$

Note that the expected quality in one deme is mP_{bb}, so the benefit of using multiple isolated demes is given by the second term of the equation above. Unfortunately, $\mu_{r:r}$ grows very slowly when r increases, and therefore the quality in the best deme is only slightly better than the quality reached by a single deme. Actually, we can easily bound $E(X_r)$ by realizing that $P_{bb}(1-P_{bb})$ is maximal when $P_{bb} = 0.5$, so

$$\hat{Q} = E(X_r) \leq mP_{bb} + \frac{\mu_{r:r}}{2}\sqrt{m}. \tag{12}$$

In the remainder of the paper, we ignore the inequality of the equation above. The required probability of success per deme may be calculated as

$$\hat{P} = \frac{\hat{Q}}{m} - \frac{\mu_{r:r}}{2\sqrt{m}}. \tag{13}$$

This equation clearly shows how the required probability of success per deme is relaxed as more demes are used.

To find the deme size we use the gambler's ruin problem in a manner similar to section 2. We must now solve $\hat{P} = P_{bb} = 1 - \left(\frac{q}{p}\right)^{n_d/2^k}$ for n_d, and the result

is

$$n_d = \frac{2^k \ln(1-\hat{P})}{\ln\left(\frac{q}{p}\right)}, \tag{14}$$

which can be approximated in terms of the domain signal and noise as

$$n_d = 2^{k-1} \ln(1-\hat{P}) \sqrt{\pi m'} \frac{\sigma_{bb}}{d}. \tag{15}$$

This equation is similar to the population size of the simple GA found in section 2.1. The only difference is that the quality required to succeed now ($\hat{P}$) is slightly smaller than before ($\hat{Q}/m$) (Note that $\hat{P}$ is equal to $\hat{Q}/m$ when $r = 1$).

To measure the benefit of using parallel isolated demes we can calculate the savings of execution time. The savings are usually normalized with respect to the serial execution time and expressed as the parallel speedup. To be fair, we compare the serial and parallel times required to find a solution of the same average quality (as was done by Cantú-Paz and Goldberg [9]). Like before, we assume that the time used by GA operations is negligible.

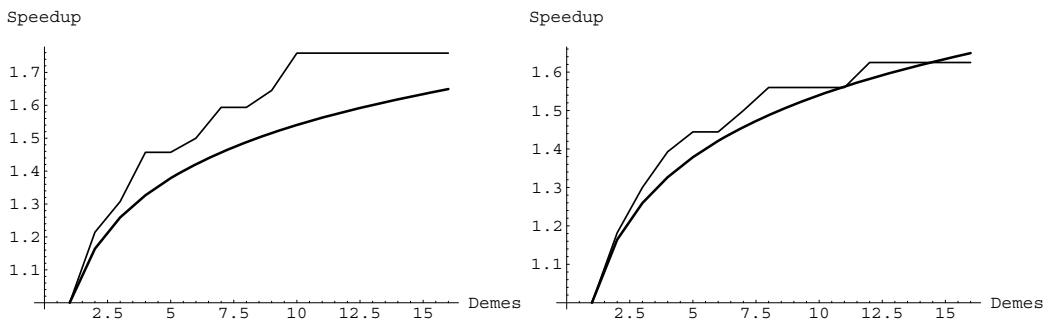
Since the demes are completely isolated, the parallel speedup is simply the ratio of the time used to evaluate individuals in the serial and parallel cases. In every generation the GA evaluates its entire population, and the number of generations needed to reach convergence may be assumed to be a domain-dependent constant, g. Holding the quality constant, the speedup for isolated demes is

$$Sp = \frac{gnT_f}{gn_dT_f} = \frac{n}{n_d}. \tag{16}$$

Substituting the population sizes for the serial and isolated parallel GAs (equations 4 and 14, respectively) into equation 16 and simplifying terms gives a simple equation for the speedup:

$$S_p = \frac{\ln(1-\frac{\hat{Q}}{m})}{\ln(1-\hat{P})}. \tag{17}$$

The accuracy of this equation was verified with experiments using the two



(a) 4-bit trap function with $m = 20$ BBs.

(b) 8-bit trap function with $m = 10$ BBs.

Fig. 6. Predicted and experimental speedups for deceptive trap functions using 1 to 16 isolated demes. The thin line shows the experimental results and the thick line is the theoretical prediction. The quality demanded in both cases was to find 80% of correct BBs ($\hat{Q} = 16$ and 8, respectively).

deceptive trap functions. The demes used pairwise tournament selection, the crossover probability was set to 1.0 and the mutation probability was zero. As before, two-point and one-point crossover were used for the 4-bit and 8-bit trap, respectively. For each deme count ($r = 1, 2, ..., 16$), the deme size was increased until at least one of the demes found a solution with at least 80% of correct BBs. At this point, we recorded the number of function evaluations. The results shown are the average over 100 independent runs. The theoretical predictions for the parallel speedup along with the experimental results are plotted in figure 6. It is evident from the figure that there is only a modest advantage in using more isolated demes, and in practice this bounding case should be avoided.

4.2 Fully Connected Demes

Executing demes in complete isolation is clearly a bound for parallel GAs, because the migration rate is zero and there are no connections between the demes. This section examines the opposite extreme of the connectivity and migration rate spectrum, where every deme exchanges individuals with all the others and the migration rate is set to its maximum value.

The analysis considers that migration occurs after the demes have converged (i.e., all the individuals in each deme are identical). At this point, each deme divides its population into r mutually exclusive sets, and exchanges the individuals of the i-th set with the i-th deme. After migration the demes restart,

and they execute until they converge again.

The time to the first convergence of each deme is referred to as the first *epoch* of the parallel GA. The second epoch starts after migration and finishes when each deme converges again. This section considers only the first two epochs of the algorithm, because a different study that analyzed the quality after several epochs found that the greatest improvement in quality comes after the second epoch [7].

The communication costs are very low when individuals migrate after convergence because communications are very infrequent, and because it is sufficient to send one individual and replicate it as necessary at the receiving deme. Also, migration after (or near) convergence has been shown empirically to have a similar effect on the solution quality as more frequent migrations [16,20].

Recall that the probability that a partition converges to the correct value in a GA with a single population is predicted by the solution to the gambler's ruin problem:

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

where x_0 is the expected number of BBs in the partition. When the demes converge the first time, a partition either has n_d copies of the correct value with probability P_{bb}, or it has n_d copies of the wrong value with probability $1 - P_{bb}$. After convergence each deme receives n_d/r migrants from every other deme. Some of those migrants contain the wrong value, but some have the correct one. Therefore, after migration each partition has on average $n_d P_{bb}$ copies of the correct value, which means that the second epoch starts from a better starting point than the first. To calculate the probability that the deme converges correctly the second time the gambler's ruin model may be used again; we only need to replace x_0 with the new starting point as follows:

$$P_{bb_2} = 1 - \left(\frac{q}{p}\right)^{n_d P_{bb}}. \tag{18}$$

Calculating this probability is the critical part of the analysis, now we can use it to find the expected number of partitions correct in the best deme as

$$E(X_r) = mP_{bb_2} + \mu_{r:r}\sqrt{mP_{bb_2}(1 - P_{bb_2})}. \tag{19}$$

There are two benefits from using multiple connected demes. As in the isolated case, the expected quality of the best deme increases by a factor of $\mu_{r:r}$, but we saw before that this is not a big difference. The major improvement comes from the expected quality in each deme (mP_{bb_2}) which is much greater than in the simple GA or the isolated demes, because the second epoch begins from a better starting point.

The required deme size for the fully connected topology will be calculated next using a procedure similar to the one used for the isolated case. The first step is to calculate the relaxation of the quality required in each deme so that the best of the demes reaches the desired solution. This was already done in the previous section, and the result is denoted by $\hat{P}$ (equation 13). The second step is to solve $\hat{P} = P_{bb_2}$ for the deme size, n_d.

We would like to use equation 18, but the P_{bb} in the exponent depends on n_d, making it impossible to obtain a closed-form expression for n_d. Instead, equation 18 has to be approximated and the deme size will be derived from the approximation. Equation 18 can be rewritten exactly as: $P_{bb_2} = 1-(1-c)^{n_d P_{bb}}$ where $c = 1 - q/p$, and in this form P_{bb_2} can be approximated as

$$P_{bb_2} \approx 1 - \exp(-cn_d P_{bb}). \tag{20}$$

Similarly, P_{bb} can be rewritten exactly as $P_{bb} = 1-(1-c)^{n/2^k}$ and approximated as $P_{bb} = 1 - e^{-cn/2^k}$. Using the first two terms of the Maclaurin series for $e^{-x} = 1 - x$, P_{bb} can be approximated roughly as $P_{bb} \approx \frac{cn}{2^k}$. Substituting this form of P_{bb} into equation 20 results in

$$P_{bb_2} \approx 1 - \exp\left(\frac{-c^2 n_d^2}{2^k}\right). \tag{21}$$

With this form of P_{bb_2} we can now solve $\hat{P} = P_{bb_2}$ to find a deme sizing equation for a fully connected topology:

$$n_d = \frac{\sqrt{-2^k \ln(1-\hat{P})}}{1 - \frac{q}{p}}, \tag{22}$$

which can be approximated in terms of the domain signal and noise as

$$n_d = \sqrt{-2^k \ln(1-\hat{P})\pi m'}\frac{\sigma_{bb}}{2d}.$$

The deme size depends on the *square root* of $2^k \ln(1-\hat{P})$ as opposed to the case for isolated demes where the deme size is directly proportional to this term. This shows that a parallel GA with migration needs smaller demes, which in turn represent a substantial reduction of the time the GA uses in computations. However, the cost of communications will partially offset the reduction in computation time.

The time that each deme uses to communicate with its $r-1$ neighbors is $(r-1)T_c$, and the time used in computations is gn_dT_f, where g represents the number of generations until convergence, n_d is the deme size, and T_f is the time required to evaluate each individual. Therefore, the execution time of the parallel program is

$$T_p = (r-1)T_c + gn_dT_f. \tag{23}$$

At this point we can observe that as r grows, there is a tradeoff between increasing communication costs and decreasing computations (because smaller demes are needed). The rest of this section deals with finding the optimal number of demes that minimizes the execution time.

To optimize the parallel time, we set $\frac{\partial T_p}{\partial r}$ to zero and solve to obtain the optimal number of demes. But, unfortunately, the derivative cannot be solved analytically for r, and to make progress in the optimization of speedups we need to characterize the parallel deme size with a simpler expression.

We observed that plotting the parallel deme size against the number of demes on a log-log scale results in an almost straight line, which means that the deme size can be approximated closely with a general power-law equation: $n_d = Ar^B$, where A and B are domain-dependent constants. The value of B may be computed as

$$B = \frac{\ln(n_1/n_2)}{\ln(r_1/r_2)},$$

where r_1 and r_2 are two arbitrary deme counts, and n_1 and n_2 are the corresponding deme sizes (obtained by using r_1 and r_2 in equation 22). The value for A can be obtained directly as $A = \frac{n_1}{r^B}$.

With this characterization of the parallel deme size, the computation time may be approximated as $gn_dT_f \approx gAr^BT_f$. Substituting this approximation into equation 23 results in

$$T_p = (r-1)T_c + gAr^BT_f.$$

Now we can set $\frac{\partial T_p}{\partial r} = 0$, and solve for r to obtain the optimal number of

demes as

$$r^* = \left(-\frac{gABT_f}{T_c}\right)^{\frac{1}{1-B}}, \tag{24}$$

and the optimal deme size is $n_d^* = Ar^{*B}$.

Figure 7 shows the predictions and the experimental results of the parallel speedups using 4- and 8-bit trap functions, respectively. The parameters for the GAs were the same as for the experiments with isolated demes, and the same experimental procedure was used. The communication time was the same as the master-slave experiments, $T_c = 19$ms, and the evaluation time for the 4- and 8-bit trap functions was $T_f = 34\mu s$ and $61\mu s$, respectively.

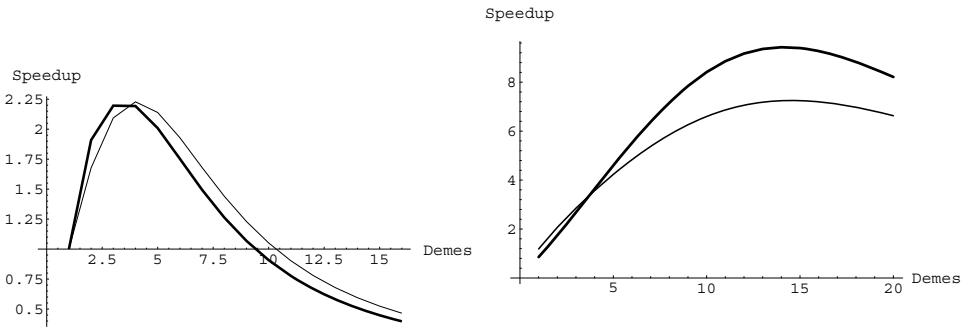
Both the predictions and the experimental results show an optimal speedup at approximately the same number of demes. Note that for the 4-bit trap function, the speedup is less than one for several deme counts, which means that the parallel algorithm actually takes longer to finish than the serial GA. This poor performance is mainly due to the expensive communications in our system, but also because the 4-bit problem can be solved with relatively small populations, and the savings on computations are too small to compensate for the rapidly increasing costs of communications.

In the case of the 8-bit trap function, the magnitude of the speedups is always greater than one for the range of demes used in the experiments. In fact, the parallel algorithm shows a significant advantage over the serial GA, even in our implementation with expensive communications. For this function the deme sizes required to find the required solution are much larger than for the 4-bit function, and the savings on the computation cost are enough to compensate for the expensive communications.

5 Hierarchical Parallel Algorithms

In previous sections we have seen that the two basic forms of parallel GAs can quickly reach good solutions when they are designed properly, but the analysis also uncovered their limitations. In particular, the analysis of the master-slave GA showed that this form of parallelism can only benefit problems with long evaluation times because it needs constant communication. On the other hand, only a little communication between demes results in much faster multiple-deme parallel GAs, but communication still imposes a limit on how fast they can be.

To overcome the inherent limits of these two basic forms of parallel GAs we can combine them into a hierarchical algorithm. The idea is simple: use a multiple-



(a) 4-bit trap function with $m = 20$ BBs.

(b) 8-bit trap function with $m = 10$ BBs.

Fig. 7. Predicted and experimental speedups for fully deceptive trap functions using 1 to 16 fully connected demes. The quality demanded was 80% of BBs correct. The thick line is the theoretical prediction and the thin line is the experimental results.

deme algorithm where the demes themselves are some form of parallel GAs. At the upper level the algorithm can be designed with the theory presented in section 4. At the lower level, we have two choices: use a master-slave GA or use a multiple-deme GA with very high migration to force panmixia.

Figure 8 depicts a hierarchical parallel GA with master-slave demes, which is the only method that has been implemented (e.g., [4]). To implement this algorithm, the number of demes and their size is determined first, and then the optimal number of slaves per deme can be determined with equation 8.

Interestingly, Goldberg invented a very similar algorithm as an object-oriented implementation of a "community model" parallel GA [13]. In each "community" (deme) there are multiple houses (slaves) where parents reproduce and the offspring are evaluated. Also, there are multiple communities and it is possible that individuals migrate to other places.

Hierarchical implementations can reduce the execution time more than any of their components alone. For example, consider that in a particular domain the optimal speedup of a master-slave GA is Sp_{ms}, and the optimal speedup of a multiple-deme GA is Sp_{md}. The overall speedup of a hierarchical GA would be $Sp_{ms} \times Sp_{md}$, and the number of processors required would be $r^* \times \mathcal{P}^*$.

The complications arise when there are not enough processors available. To minimize the execution time, the processors have to be allocated in the most efficient way, and the best combination of demes/slaves has to be found. One alternative is simply to enumerate all possible pairs, calculate the expected speedup for each, and choose the pair with the highest speedup. Actually, the number of pairs is not too large: consider that the number of processors $\mathcal{P}$

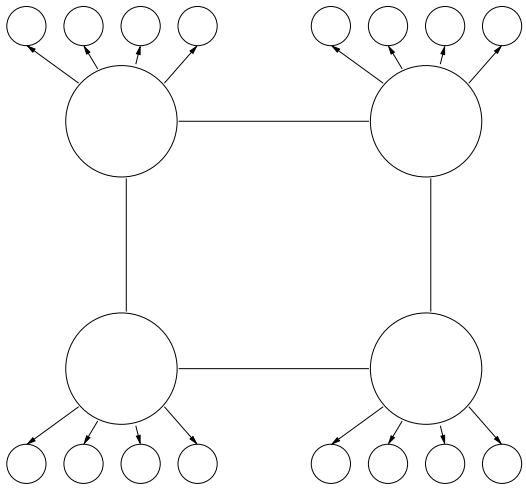


Fig. 8. A schematic of a hierarchical parallel GA. At the upper level this hybrid is a multi-deme parallel GA where each node is a master-slave GA.

is constrained by $r(S+1) \leq \mathcal{P}$, so the number of slaves for a particular r is constrained as $s \leq \mathcal{P}/r - 1$. Since the number of demes can go from 1 to $\mathcal{P}$, the number of pairs to be considered is

$$\sum_{r=1}^{\mathcal{P}} (\mathcal{P}/r - 1) = \mathcal{P} \sum_{r=1}^{\mathcal{P}} 1/r - \mathcal{P} = \mathcal{P}(\ln \mathcal{P} + O(1)) - \mathcal{P}.$$

It is likely than when the ratio of communication to computation time is high, a configuration with few slaves per deme will perform better, because there are fewer communications than in a configuration with many slaves per deme. Similarly, when the fitness evaluations are very expensive, it may be more beneficial to use more slaves per deme.

6 Implementing Scalable Multi-Population GAs

The analysis of the two bounding cases resulted in useful design guidelines for parallel GAs, but those cases are not commonly used by practitioners because neither of them is scalable. On one hand, when the demes are isolated the total computational cost grows very fast. On the other extreme, the cost of communicating with every other deme can quickly become impractical as the number of demes grows, even considering that in our algorithm communications are sparse because they only occur after the demes converge.

This section addresses the problem of scalable communications by analyzing the effects of the neighbors of each deme, the deme size, and the migration rate on the cost and on the solution's quality. The analysis shows that there is

an optimal deme size and a number of neighbors that together minimize the execution time of the parallel GA.

Recall that the critical part of the model for fully connected demes was to use the expected number of BBs as the starting point of the second epoch. The problem with this approach is that *on average* the number of BBs is the same at the start of the second iteration (i.e., $n_d P_{bb}$), regardless of the migration rate and of how the demes are connected, and therefore the model cannot be used to determine the likelihood of reaching the solution.

An important property of the connectivity graph between the demes is its *degree*, which is the number of neighbors of each deme, and we denote it as δ. The degree completely determines the communications cost, and it also affects the reliability of the algorithm. As before, the analysis in this section considers only the first two epochs of the algorithm.

The analysis has several steps. First, we compute how many copies of the correct BB are necessary to reach the target quality. Next, we calculate the probability that a given configuration brings together the critical number of BBs. Then, the success probability is used to derive a deme sizing equation, which in turn is used to minimize the execution time.

Recall that the gambler's ruin model relates the population size and the number of BBs present initially to the solution quality. To find how many BB are needed at the start of the second epoch to reach the target quality $\hat{P}$ we simply solve the equation that predicts convergence to the right BB for $\widehat{x_1}$:

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

which results in

$$\widehat{x_1} = \frac{\ln(1-\hat{P})}{\ln\left(\frac{q}{p}\right)}. \tag{25}$$

The next step in the analysis is to determine the probability that a deme receives at least $\widehat{x_1}$ BBs from its δ neighbors. The probability that a 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 $\hat{\delta} = \widehat{x_1}/(n_d\rho)$ 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:

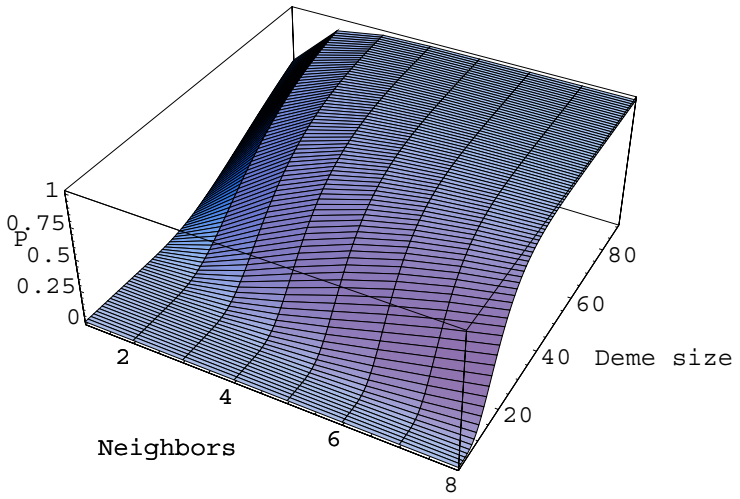


Fig. 9. Plot of the probability of success P_{x_1} (equation 27) with different configurations of deme sizes (n_d) and number of neighbors (δ).

$$P_{x_1} = 1 - \sum_{i=0}^{\hat{\delta}-1} \binom{\delta}{i} P_{bb}^{i} (1 - P_{bb})^{\delta - i}, \tag{26}$$

which can be approximated with a normal distribution as

$$P_{x_1} = 1 - \Phi \left(\frac{\hat{\delta} - \delta P_{bb}}{\sqrt{\delta P_{bb}(1 - P_{bb})}} \right). \tag{27}$$

When the migration rate is high, the number of neighbors that must contribute the right BB decreases, and therefore it is more likely that the deme receives the critical number of BBs. This observation matches other studies that showed that increasing the migration rate resulted in higher average quality per deme [7].

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 still 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} becomes 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 (see figure 9). Configurations with large demes and few neighbors have the same chance of succeeding than some configurations with smaller demes but with more neighbors. This is the usual tradeoff between computation and computations: smaller demes mean less computations, but require more neighbors to succeed. We would like to use the configuration that achieves the objective with the minimal cost.

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

$$T_p = \delta T_c + g n_d T_f, \tag{28}$$

where T_c is the time required to communicate with one neighbor, T_f is the time of one fitness evaluation, g represents the number of generations until convergence, and n_d is the deme size. 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.

6.1 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 $\Phi(z) = (1 + \exp(-1.6z))^{-1}$ [22]. With this approximation, P_{x_1} becomes

$$P_{x_1} = 1 - (1 + \exp(-1.6z))^{-1}, \tag{29}$$

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. In addition, P_{bb} may be roughly approximated as $P_{bb} \approx \frac{cn}{2^k}$, where $c = 1 - q/p$ (see the discussion before equation 21). 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 $z = \hat{z}$, solving for n_d, and simplifying terms gives the deme size:

$$n_d = \frac{2^{k-2}}{\sqrt{\delta}} \frac{\hat{z} + \sqrt{\hat{z}^2 + \frac{c\widehat{x_1}}{\rho 2^{k-2}}}}{c}. \tag{30}$$

Observe that the deme size decreases with higher migration rates and as the

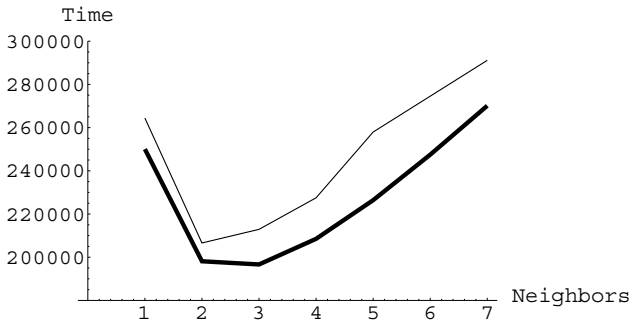


Fig. 10. Comparison of theoretical (thick line) and experimental (thin line) execution times (in microseconds).

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:

$$n_d = \frac{n_0}{\sqrt{\delta}}. \tag{31}$$

Now, the total execution time as given by equation 28 may be easily optimized with respect to delta by making $\frac{\partial T_p}{\partial \delta} = 0$ and solving for δ:

$$\delta^* = \left(\frac{g n_0 T_f}{2 T_c} \right)^{2/3}, \tag{32}$$

and the optimal deme size can be found by substituting δ^* in equation 30.

Figure 10 compares the theoretical predictions of the execution time with experimental results on a network of eight IBM workstations. 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 $g = 50$. The migration rate is set to 10%.

7 Summary and Conclusions

To design fast and reliable parallel GAs one must decide on a configuration among the many choices of topologies, migration rates, number and size of demes. Each parameter affects the quality of the search and the efficiency of the algorithm in non-linear ways, and our goal is to determine the configuration that finds a solution of the required quality as fast as possible.

Our investigation of parallel GAs centers on sizing populations correctly, because previous studies highlighted the crucial relation between population size and solution quality. Therefore, our study began with a review of population sizing for simple GAs. Then, we examined what is possibly the simplest implementation of a parallel GAs as a synchronous master-slave algorithm to determine a lower bound on the potential speedups of parallel GAs. The analysis showed that even simple implementations may result in substantial time savings, and it described how to minimize the execution time.

Next, we examined bounding cases of multiple-deme GAs. The calculations showed only very modest speedups when the demes remain isolated, and therefore this extreme should be avoided in practice. But, when the demes communicate there are important reductions on the computation time. The analysis determined the optimal number of demes and the corresponding deme size that minimize the execution time.

Finally, the deme sizing equations were specialized to consider the relationships between deme size, migration rate, and the topology's degree with the probability of success. As before, the result of the analysis was a configuration that optimizes the execution time, while still reaching the desired target solution.

This paper is an important contribution to the design of efficient, accurate, and reliable parallel GAs. Theoreticians and practitioners alike can benefit from the results of this study. On one hand, theoreticians may adopt the decomposition of the complex design problem into semi-independent facets to obtain simple, useful, and accurate models of other aspects of (parallel) GAs. Our works stands as a proof of principle that exact models are not necessary to grasp the effects of the problem's difficulty and the parameters of parallel GAs on their quality and efficiency.

On the other hand, practitioners no longer need to use blind guessing or expensive systematic experimentation to determine the size and number of populations or how to connect them. Instead, they can use the tools presented here to choose a configuration that yields good results fast and reliably. Some experimentation is necessary to calibrate the models to the particular domain and hardware environment, but these experiments are cheap and easy to perform.

Acknowledgments

This study was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants number F49620-94-1-0103,

F49620-95-1-0338, and F49620-97-1-0050. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

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

We would like to thank the anonymous reviewers whose comments helped us to improve the quality of this paper.

References

[1] M. Abramovitz and I. Stegun, editors. *Handbook of mathematic functions with formulas, graphs, and mathematical tables*. Dover Publications, New York, 1972.

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

[3] A. D. Bethke. 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.

[4] R. Bianchini and C. Brown. Parallel genetic algorithms on distributed-memory architectures. *Transputer Research and Applications*, 6:67–82, 1993.

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

[6] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallèles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.

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

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

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

[10] K. Deb and D. E. Goldberg. Analyzing deception in trap functions. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 93–108, San Mateo, CA, 1993. Morgan Kaufmann.

[11] W. Feller. *An Introduction to probability theory and its applications*, volume 1. Wiley, 2nd edition, 1966.

[12] T. C. Fogarty and R. Huang. Implementing the genetic algorithm on transputer based parallel processing systems. *Parallel Problem Solving from Nature*, pages 145–149, 1991.

[13] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.

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

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

[16] P. B. Grosso. *Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model*. PhD thesis, The University of Michigan, 1985.

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

[18] H. Leon Harter. *Order Statistics and Their Use in Testing and Estimation*. U.S. Government Printing Office, Washington, D.C., 1970.

[19] R. Hauser and R. Männer. Implementation of standard genetic algorithm on MIMD machines. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving fron Nature, PPSN III*, pages 504–513, Berlin, 1994. Springer-Verlag.

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

[21] D. Thierens and D. E. Goldberg. Mixing in genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45, San Mateo, CA, 1993. Morgan Kaufmann.

[22] M. Valenzuela-Rendón. *Two analysis tools to describe the operation of classifier systems*. PhD thesis, University of Alabama, Tuscaloosa, 1989.