Designing Efficient and Accurate Parallel Genetic Algorithms

Erick Cantú-Paz

IlliGAL Report No. 99017
July 1999

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Matthews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

DESIGNING EFFICIENT AND ACCURATE PARALLEL GENETIC ALGORITHMS

BY

ERICK CANTÚ-PAZ

Ingen., Instituto Tecnológico Autónomo de México, 1994

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1999

Urbana, Illinois

Abstract

Parallel implementations of genetic algorithms (GAs) are common, and, in most cases, they succeed to reduce the time required to find acceptable solutions. However, the effect of the parameters of parallel GAs on the quality of their search and on their efficiency are not well understood. This insufficient knowledge limits our ability to design fast and accurate parallel GAs that reach the desired solutions in the shortest time possible. The goal of this dissertation is to advance the understanding of parallel GAs and to provide rational guidelines for their design. The research reported here considered three major types of parallel GAs: simple master-slave algorithms with one population, more sophisticated algorithms with multiple populations, and a hierarchical combination of the first two types. The investigation formulated simple models that predict accurately the quality of the solutions with different parameter settings. The quality predictors were transformed into population-sizing equations, which in turn were used to estimate the execution time of the algorithms. The primary tradeoff between decreasing computations and increasing communications was identified, and it was used to find the optimal configuration of each algorithm that minimized the execution time. The investigation is mainly theoretical, but experimental evidence using test functions of varying difficulty is included to illustrate the accuracy of the theory. The results of this investigation enable practitioners to determine what algorithm is the most beneficial for their particular domain and to allocate the resources available in the most efficient manner.

For Javiera

Acknowledgments

Undoubtedly, the person who most influenced the research reported here is my advisor, David E. Goldberg. I wish to thank him for his guidance, support, and encouragement during my time at the University of Illinois. Besides all the technical and methodological issues, I learned from him much about the "human side" of engineering.

I wish to thank Prof. Geneva Belford, Prof. Sylvian Ray, and Prof. Josep Torrellas for their comments on my work and for their patience when delays occurred. Over my stay in Illinois, I had the opportunity to learn many different things as I worked with each of them.

I was very fortunate to be surrounded by a very productive and cheerful group of people in the Illinois Genetic Algorithms Lab. I wish to thank all the students, visitors, and librarians at IlliGAL for their friendship and for all the useful discussions.

I thank the numerous reviewers that made useful comments on parts of this dissertation that were submitted to conferences and journals. Their comments helped to make a clearer exposition of some of the ideas contained herein.

My sisters and I were blessed with wonderful parents who gave us the perfect example of how to live a happy life. No words can express my gratitude for everything that they have done; I can only hope to be able to transmit their wisdom to my own children.

Finally, I thank my wife, Javiera, for all her love and understanding. Her support kept me going during hard and frustrating times. I could not have finished this project without her.

My research was sponsored by a number of sources: a Fulbright-García Robles Fellowship, administered by USIA and by CONACyT; a teaching assistantship from the Department of Computer Science, UIUC; and a research assistantship from the Beck-

man Institute, UIUC. The work 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. 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.

Table of Contents

Chapter

List of Tables

List of Figures

Chapter 1

Introduction

To solve some problems, genetic algorithms (GAs) may require hundreds or thousands of function evaluations. Depending on the cost of each evaluation, the GA may take days, months, or even years to find an acceptable solution. Fortunately, GAs work with a population of independent solutions, which makes it easy to distribute the computational load among several processors. For a long time, the parallel nature of genetic algorithms has been recognized, and many have successfully used parallel GAs to reduce the time required to reach adequate solutions to complex problems.

Indeed, some could say that GAs are "embarrassingly parallel" programs, and that it is trivial to make fast parallel GAs. But despite their operational simplicity, parallel GAs 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 or populations must be determined carefully, and when multiple populations are used, one must decide how many to use. In addition, the populations may remain isolated or they may communicate by exchanging individuals. 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 systematic experimentation or are found just by chance. The problem with these approaches is that they often result in a waste of computing resources or in an inadequate search quality, and practitioners

may regard parallel GAs as being impractical or unreliable.

The goal of this investigation is to go beyond this ad hoc tuning of parameters, and provide guidelines to choose their values rationally. The result are efficient parallel GAs that consistently reach solutions of high quality.

The research methodology combines theory and experiments to develop equations that capture the effect of the parameters on the speed and quality of the algorithm. However, this study does not develop exact models of every facet of (parallel) genetic algorithms, because exact and complete models would probably be too difficult to use, and therefore would have little practical significance. Instead, the intention is to create small easy-to-calibrate models that serve as a solid basis to design parallel GAs.

The research focuses on sizing the populations correctly, because previous studies have shown that the size of the population greatly determines the quality of the search (Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997) and its duration (Goldberg & Deb, 1991).

1.1 Organization

This dissertation is organized in three parts, which correspond to the decomposition used in the research. The basic approach in each part is to determine first how the configuration of the algorithm affects the quality of the solutions. Then, the models of quality are transformed into population-sizing equations that incorporate the domain-dependent parameters of interest. Finally, the primary tradeoffs are identified and used to optimize the execution time of the algorithms.

The first part of the dissertation studies GAs with a single population. Chapter 2 presents an accurate model that relates the population size and the problem difficulty with the solution quality of a simple GA. The model is built by making an analogy between selection in GAs and biased random walks. It integrates previous

knowledge about correct decision-making at the building block level with knowledge of the expected number of building blocks in the initial random population.

Chapter 3 shows how to parallelize GAs using a simple master-slave algorithm. The analysis shows how to minimize the execution time, but most importantly, it gives a lower bound on the acceptable performance of parallel GAs. Other parallel GAs should perform better than the simple case examined there.

Most of this project focuses on multi-population algorithms, because they are the most popular, the most complex, and potentially the most efficient, because they require fewer communications than the master-slave. However, to design multiple-deme parallel GAs one must face difficult and interrelated choices. The three main problems are to determine (1) the size and the number of demes (subpopulations), (2) the topology that interconnects the demes, and (3) the migration rate that controls how many individuals migrate.

It is difficult to study all these problems at once, and therefore, the second part of the dissertation considers two bounding cases of multi-population GAs. In the first case the populations evolve in complete isolation, which is a lower bound on the migration rate and on the connectivity between the demes. The upper bound occurs when the demes are fully connected and they exchange a maximum number of individuals. The goal of the investigation of bounding cases is to make some progress on the deme-sizing issue without ignoring the topology and migration rates. Usually, practitioners would not use these extreme cases (especially the fully connected topology), but the analysis sheds some light on the intermediate configurations, and establishes the methodology that will be used later.

The study of bounding cases spans three chapters. Chapter 4 shows how to extend the model for the simple GA to predict the expected quality for each bounding case. The analysis reveals that the size of the parallel demes may be reduced, while maintaining a constant target solution quality. The reduction in the deme sizes results in shorter execution times. Chapter 5 quantifies the gains in performance,

and shows that there is an optimal configuration that minimizes the execution time. This chapter also discusses the controversial claims of superlinear speedups in parallel GAs, and explains possible causes for the reduction of the total work performed by the parallel populations. Chapter 6 uses Markov chains to analyze the long-term behavior of communicating demes. Initially, it studies the fully connected case, and it shows the effect of varying the migration rate on the quality of the solutions.

Chapter 6 also introduces the first model for intermediate topologies, which are the topic of the third part of the dissertation. This first model is very accurate, but it requires considerable computations, and therefore may not be practical to study large numbers of demes. Nevertheless, the model shows that the topology and the migration rate greatly influence the solution quality, and it is used to verify the accuracy of the simpler models derived in the next chapter.

Chapter 7 builds simple approximate models that capture the effects of the topology and migration rate on the quality and cost of multi-deme GAs. Those models are then used to find the settings for the communication parameters that minimize the execution time. This chapter also shows that an optimally-configured fully-connected topology may competitive, because it uses few demes to obtain substantial time reductions.

At this point, the dissertation has investigated the benefits of master-slave and multiple-deme parallel GAs. Practitioners may use the theory presented in the dissertation to determine what algorithm delivers the greatest benefit for their problem. However, the choice is not so straightforward because master-slaves and multiple-deme GAs may be combined into hierarchical algorithms that combine the benefits of their components. Chapter 8 describes how to allocate the available processors between demes and slaves to minimize the overall execution time. This chapter also contains a complete step-by-step example that illustrates how to determine the optimal configuration of a parallel GA for a particular problem domain and hardware environment.

Finally, chapter 9 contains a summary of the results, recommendations for further research, and the conclusions of this study.

Chapter 2

The Gambler's Ruin Problem and the Sizing of Populations

The population of a genetic algorithm is formed by individuals with artificial chromosomes that encode solutions of a 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 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.

One of the most important decisions needed to use a GA is to set the number of individuals in the population. However, the question of how to choose an adequate population size for a particular domain is difficult, and has puzzled GA practitioners for a long time. 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 large, the GA will waste time processing unnecessary individuals, and this may result in unacceptably slow performance. The problem consists on finding a population size that is large enough to permit a correct exploration of the search space, and that does not waste computational resources. The same problem is

encountered in parallel GAs, and therefore the results of this chapter are important to our quest for efficient parallel GAs.

This chapter presents a model to predict the convergence quality of genetic algorithms based on the size of their population. The model is based on an analogy between selection in GAs and one-dimensional random walks. Using the solution to a classic random walk problem (the gambler's ruin), the model naturally incorporates previous knowledge about the initial supply of building blocks and the correct selection of the best BB over its competitors. The result is an accurate relation between the size of the population with the desired quality of the solution, as well as the problem size and difficulty.

The chapter is largely based on a paper by Harik, Cantú-Paz, Goldberg, and Miller (1997), and is organized as follows. The next section presents the facetwise decomposition that guides this work and a discussion of some previous studies on population sizing. Section 2.2 revisits in detail a model that describes the probability of choosing correctly between two competing individuals. This pairwise decision probability is then used as an integral part of the population sizing model presented in section 2.3. Then, section 2.4 confirms the accuracy of the model with results of experiments based on domains of varying difficulty. Section 2.5 discusses how the model is extended to account for explicit noise in the fitness of the individuals, and section 2.6 extends the model to consider different selection types. The chapter concludes with a brief summary.

2.1 Background

Over the years, researchers and practitioners have noticed that the population size largely determines the convergence quality of GAs and the duration of their run. Unfortunately, there are only a handful of studies that guide users to choose adequate population sizes (Goldberg, 1989b; Goldberg, Deb, & Clark, 1992). This section

reviews some of these studies, but first it describes the decomposition that guides our study of GAs.

2.1.1 Decomposing the Problem

Despite their operational simplicity, GAs are complex non-linear algorithms. To have any hope of understanding and designing GAs we need to approach them as we do with other difficult engineering tasks: decompose the problem into tractable sub-problems, solve the sub-problems, and integrate the partial solutions. For some time now, we have used a rational, facetwise decomposition as a guide in our study of GAs (Goldberg, 1991; Goldberg, Deb, & Clark, 1992):

1. Know what the GA is processing: building blocks (BBs)
2. Solve problems tractable by BBs
3. Supply enough BBs in the initial population
4. Ensure the growth of necessary BBs
5. Mix the BBs properly
6. Decide well among competing BBs

We restrict the notion of building blocks to the minimal-order schemata that contribute to the global optimum. Other schemata in the same partition—even if they have a high average fitness—are labeled as incorrect. In this view, when crossover juxtaposes two BBs of order k at a particular string it does not lead to a single BB of order $2k$, but instead to two separate BBs (Thierens & Goldberg, 1993). In addition, we assume that the only source of BBs is the random initialization of the population: mutation and crossover do not create or destroy too many BBs.

This study involves two of the six points above: the initial supply and the decision process between competing BBs. The model described later in this chapter

incorporates these two issues in what elsewhere (Goldberg, 1996b) has been called a *little model*; it does not attempt to describe the effect of all possible parameters on the search. Instead, the model focuses only on the supply and decision issues and describes many practical relations between them. Although the model excludes the effects of mixing and growth of BBs, the result is extremely accurate and can be used as a guideline to design faster and more reliable GAs.

Previous estimates of adequate population sizes consider only one of these facets, while the model introduced later in this chapter naturally integrates both. The remainder of this section reviews some of these previous studies.

2.1.2 Supply Models

Before selection and recombination can act on the BBs, the GA first needs to have an adequate supply. When BBs are abundant, it is likely that the GA will choose and combine them correctly; conversely, when BBs are scarce the chances of the GA converging to a good solution are low.

The first supply model simply considers the number of BBs present in the initial random population. The probability that a single building block of size k is generated randomly is $1/2^k$ for binary domains, and therefore the initial supply of BBs can be estimated as

$$x_0 = \frac{n}{2^k}. \tag{2.1}$$

This simple supply equation suggests that domains with short BBs—and thus with more BBs in the initial population—need smaller population sizes than domains with longer BBs. A later section presents empirical results using functions with BBs of different lengths to corroborate this notion.

Another way of relating the size of the population with the expected performance

of the GA is to count the number of schemata processed by the GA. Holland (1975) estimated that a randomly initialized population of size n contains $O(n^3)$ schemata. Holland used the term *implicit parallelism* to denote this fact, and it has become one of the common arguments on why GAs work well. Goldberg (1989a) rederived this estimate in two steps: (1) compute the number of schemata in one string, and then (2) multiply it by the population size.

The number of schemata of length l_s or less in one random binary string of length l is $2^{l_s-1}(l-l_s+1)$. The schema length l_s is chosen such that the schemata survive crossover and mutation with a given constant probability. It is likely that low-order schemata will be duplicated in large populations, so to avoid overestimating the number of schemata, pick a population size $n = 2^{l_s/2}$, so that on average half of the schemata are of higher order than $l_s/2$ and half are of lower order. Counting only the higher order ones gives a lower bound on the number of schemata in the population as $n_s \geq n(l-l_s+1)2^{l_s-2}$. Since $n = 2^{l_s/2}$ this becomes $n_s = \frac{(l-l_s+1)n^3}{4}$, which is $O(n^3)$.

In a different study, Goldberg (1989b) computed the expected number of unique schemata in a random population, and used this quantity together with an estimate of the convergence time to find the optimal population size that maximizes the rate of schema processing. Goldberg considered serial and parallel fitness evaluations, and his results suggest that high schema turnover is promoted with small populations in serial GAs and with large populations in parallel.

More recently, Mühlenbein and Schlierkamp-Voosen (1994) derived an expression for the minimum population size needed to converge to the optimum with high probability. Their analysis is based on additive fitness functions, and their study focuses on the simplest function of this type: the one-max, which we also use in our investigation. They conjectured that the optimal population size depends on the initial supply of the desired alleles, on the size of the problem, and on the selection intensity. Our study also considers the effect of these variables on the population size. Later, Cvetković and Mühlenbein (1994) empirically determined that for the

one-max the population size is directly proportional to the square root of the size of the problem, and it is inversely proportional to the square root of the proportion of correct alleles in the initial population. Our results are also based on additive fitness functions, and are consistent with their experimental fit on the problem size, but indicate that in general the population size is inversely proportional to the proportion of correct BBs present initially (not to the square root).

2.1.3 Decision Models

The second aspect of population sizing involves selecting better partial solutions. Holland (1973, 1975) recognized that the issue of choosing between BBs (and not between complete strings) can be recast as the two-armed bandit problem, a well-known problem in statistical decision theory. The problem consists in choosing the arm with the highest payoff of a two-armed slot machine, at the same time that payoff information is collected. This classic problem is a concrete example of the tradeoff between exploring the sample space and exploiting the information already gathered. Holland's work assumes an idealization of the GA as a cluster of interconnected 2-armed bandits, so his result relating the expected loss and the number of trials can be directly applied to schema processing. Although Holland's calculations are based on an idealization, his results give an optimistic bound on the allocation of trials on a real GA.

Around the same time, De Jong (1975) recognized the importance of noise in the decision process and proposed an estimate of the population size based on the signal and noise characteristics of the problem. Unfortunately, he did not use his estimate in the remaining of his groundbreaking empirical study and the result remained unverified and ignored by many.

Goldberg and Rudnick (1991) gave the first population sizing estimate based on the variance of fitness. Later, Goldberg, Deb, and Clark (1992) developed a conservative bound on the convergence quality of GAs. Their model is based on

deciding correctly between the best BB in a partition and its closest competitor, while the decision process is clouded by collateral noise coming from the other partitions.

The result of that investigation is a population sizing equation that for binary alphabets and ignoring external sources of noise is:

$$n = 2c(\alpha)2^k m' \frac{\sigma_M^2}{d^2}, \tag{2.2}$$

where $c(\alpha)$ is the square of the ordinate of a unit normal distribution where the probability equals α; α is the probability of failure; k is the order of the BB; m' is one less than the number of BBs in a string (m); σ_M^2 is the average fitness variance of the partition that is being considered; and d is the fitness difference between the best and second best BBs.

This model conservatively approximates the behavior of the GA by considering that if the wrong BBs were selected in the first generation, the GA would be unable to recover from the error. Likewise, if the decisions were correct in the first generation, the model assumes that the GA would converge to the right solution. This study is a direct extension of the work by Goldberg, Deb, and Clark. The main difference between the current model and theirs is that we do not longer approximate the behavior of the GA by the outcome of the first generation.

2.2 Deciding Well Between two BBs

The role of selection in GAs is to decide which individuals survive to form the next generation. The selection mechanism is supposed to choose those individuals that have the correct BBs and to eliminate the others, but sometimes the wrong individuals are chosen. To understand why this may occur this section reviews the calculations of Goldberg, Deb, and Clark (1992) of the probability of deciding well between an individual with the best BB and another individual with the second best BB in

Figure 2.1: Two competing building blocks of order four.

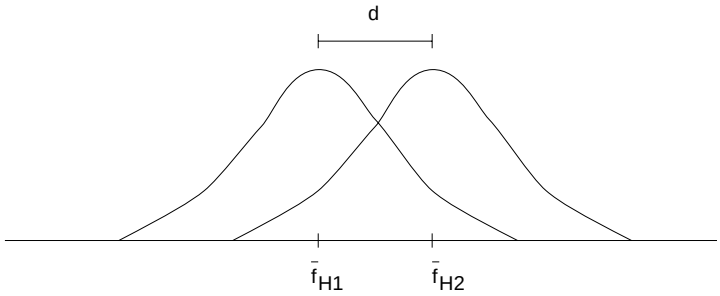


Figure 2.2: Fitness distributions for two competing individuals.

a partition. Their idea is to focus on one partition, and to consider the fitness contributions from the other partitions as noise that interferes in the decision process.

Consider 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. This is illustrated in figure 2.1. 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 $f_1 - f_2 > 0$.

Figure 2.2 illustrates the distributions of the fitness of the chromosomes containing H_1 and H_2. The distance between the mean fitness of H_1 ($\bar{f}_{H_1}$) and the mean fitness of H_2 ($\bar{f}_{H_2}$) is denoted by d. Assuming that the fitness is an additive function of the fitness contribution of all the partitions in the problem, we may consider that the fitness distributions of H_1 and H_2 to be normal by the central limit theorem (because the fitness contribution of each partition is a random variable, and the sum of random variables is normally distributed).

Since the fitness distributions of i_1 and i_2 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,

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

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

$$p = \Phi\left(\frac{d}{\sqrt{\sigma^2_{H_1} + \sigma^2_{H_2}}}\right), \tag{2.3}$$

where Φ is the cumulative distribution function (CDF) of a normal distribution with zero mean and unit standard distribution.

To calculate the variance of H_1 and H_2, we follow Goldberg, Deb, and Clark (1992), and assume that the fitness function is the sum of m independent subfunctions F_i, each of the same size k of the most deceptive partition. Now, the overall fitness variance may be calculated as

$$\sigma^2_F = \sum_{i=1}^{m} \sigma^2_{F_i}.$$

For domains where the m partitions are uniformly scaled (equally weighted), the average BB variance (denoted by σ^2_{bb}) is simply $\sigma^2_{F_i}$. In this case the total noise coming from the $m' = m-1$ partitions that are not competing directly is $\sigma^2_F = m'\sigma^2_{bb}$. Therefore, the probability of making the right choice in a single trial in a problem with m independent and equally-scaled partitions becomes

$$p = \Phi\left(\frac{d}{\sqrt{2m'}\sigma_{bb}}\right). \tag{2.4}$$

Goldberg, Deb, and Clark (1992) used this probability to create the first model that relates the size of the population with the quality of decisions. Their model showed how to incorporate the effect of collateral noise into the population-sizing question, and their paper describes how to estimate the parameters necessary to calculate p. In the next section, that knowledge about noise and decision-making at the BB level will be unified quite naturally with the knowledge about the initial supply of BBs.

2.3 The Gambler's Ruin Model

The gambler's ruin problem is a classical example of random walks, which are mathematical tools that can be used to predict the outcome of certain stochastic processes. The most basic random walk deals with a particle that moves randomly on a one-dimensional space. The probability that the particle moves to the left or to the right is known, and it remains constant for the entire experiment. The size of the step is also constant, and sometimes the movement of the particle is restricted by placing barriers at some points in the space. For our purposes, we consider a one-dimensional space bounded by two absorbing barriers that capture the particle once it reaches them.

In the gambler's ruin problem, the capital of a gambler is represented by the position, x, of a particle on a one-dimensional space, as depicted in figure 2.3. Initially, the particle is positioned at $x_0 = a$ where a represents the gambler's starting capital. The gambler plays against an opponent that has an initial capital of $n - a$, and there are absorbing boundaries at $x = 0$ (representing bankruptcy) and at $x = n$ (representing winning all the opponent's money). At each step in the game, the gambler has a chance p of increasing his capital by one unit and a probability $q = 1-p$ of loosing one unit. The object of the game is to reach the boundary at $x = n$, and the probability of success depends on the initial capital and on the probability of

Figure 2.3: The bounded one-dimensional space of the gambler's ruin problem.

winning a particular trial.

The connection between selection in GAs and the gambler's ruin problem was first proposed by Harik, Cantú-Paz, Goldberg, and Miller (1997), and appears naturally when we assume that partitions are independent and concentrate on only one of them. The particle's position on the one-dimensional space, x, represents the number of copies of the correct BBs in the population. The absorbing barriers at $x = 0$ and $x = n$ represent convergence to the wrong and 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, which in a binary domain and considering BBs of order k is $x_0 = \frac{n}{2^k}$.

There are a number of assumptions that we need to make to use the gambler's ruin problem to predict the quality of the solutions of the GA. The gambler's ruin 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, and the outcome of each decision is to win or lose one copy of the optimal BB. Assuming conservatively that all competitions occur between strings that represent the best and the second best BBs in a partition, the probability of gaining a copy of the global BB is given by the correct decision-making probability p (equation 2.4).

The calculation of p implicitly assumed that that the GA uses pairwise tournament selection (two strings compete), but adjustments for other selection schemes are possible, as we shall see in a later section. The analogy between GAs and the

gambler's ruin problem also assumes that the only source of BBs is the random initialization of the population. This assumption implies that mutation and crossover do not create or destroy significant numbers of BBs. 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. We recognize that this model is a simplification, but experimental results suggest that is it a reasonable one.

As we discussed above, the GA succeeds when there are n copies of the correct BB in the partition of interest. A well-known result in the random walk literature is that the probability that the particle will eventually be captured by the absorbing barrier at $x = n$ is (Feller, 1966):

$$P_{bb} = \frac{1 - \left(\frac{q}{p}\right)^{x_0}}{1 - \left(\frac{q}{p}\right)^{n}}. \tag{2.5}$$

From this equation, it is relatively easy to find an expression for the population size. First, note that by definition $p > 1 - p$ (because the mean fitness of the best BB is greater than the mean fitness of the second best), and that x_0 is usually small compared to the population size. Therefore, for increasing values of n the denominator in equation 2.5 approaches 1 very quickly and it can be ignored in the calculations. Substituting the initial supply of BBs ($x_0 = n/2^k$), P_{bb} may be approximated as

$$P_{bb} \approx 1 - \left(\frac{1-p}{p}\right)^{n/2^k}. \tag{2.6}$$

We measure the quality of the solutions as the number of partitions X that converge to the correct BB. Since we assume that the BBs are independent of each

other, the expected number of partitions with the correct BB at the end of a run is $E(X) = mP_{bb}$. Assuming that we are interested in finding a solution with an average of $\hat{Q}$ partitions correct, we can solve $P_{bb} = \frac{\hat{Q}}{m}$ for n and obtain a population sizing equation:

$$n = \frac{2^k \ln(\alpha)}{\ln\left(\frac{1-p}{p}\right)}, \tag{2.7}$$

where $\alpha = 1 - \frac{\hat{Q}}{m}$ is the probability of failure. To observe more clearly the relations between the population size and the domain-dependent variables involved, we may 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 (Abramowitz & Stegun, 1972):

$$p = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} z,$$

where $z = d/(\sigma_{bb}\sqrt{2m'})$. Substituting this approximation for p into equation 2.7 results in

$$n = 2^k \ln(\alpha) / \ln\left(\frac{1 - \frac{z\sqrt{2}}{\sqrt{\pi}}}{1 + \frac{z\sqrt{2}}{\sqrt{\pi}}}\right). \tag{2.8}$$

Since z tends to be a small number, $\ln(1 \pm \frac{z\sqrt{2}}{\sqrt{\pi}})$ may be approximated as $\pm\frac{z\sqrt{2}}{\sqrt{\pi}}$. Using these approximations and substituting the value of z into the equation above gives

$$n = -2^{k-1} \ln(\alpha) \frac{\sigma_{bb}\sqrt{\pi m'}}{d}. \tag{2.9}$$

This rough approximation makes more clear the relations between some of the variables that determine when a problem is harder than others. It quantifies many intuitive notions that practitioners have about problem difficulty for GAs. For example, problems with long BBs (large k) are more difficult to solve than problems with short BBs, because long BBs are scarcer in a randomly initialized population. Furthermore, the equation shows that the required population size is inversely proportional to the signal-to-noise ratio. Intuitively, problems with a high variability are hard because it is difficult to detect the signal coming from the good solutions when the interference from not-so-good solutions is high. Similarly, longer problems (larger m) are more difficult than problems with a few partitions because there are more sources of noise. However, the GA scales very well to the problem size; the equation shows that the required population grows as the square root of the size of the problem.

2.4 Experimental Verification

This section verifies that the gambler's ruin model predicts accurately the quality of the solutions reached by simple GAs. The experiments reported in this section use additively-decomposable test functions of varying difficulty, which will be used to test the parallel GA models presented in later chapters. The population sizes required to solve the test problems vary from a few tens to a few thousands, demonstrating that the predictions of the model scale well to problem difficulty.

All the results in this chapter are the average over 100 independent runs of a simple generational GA. The GA uses pairwise tournament selection without replacement. The crossover operator was chosen according to the order of the BBs of each problem, and is specified in each subsection below. In all the experiments, the mutation probability is set to zero, because the model considers that the only source of diversity is the initial random population. 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 converged to the correct value. The theoretical predictions of the gambler's ruin model were calculated with equation 2.5.

2.4.1 One-max

The one-max problem is probably the most frequently-used fitness function in genetic algorithms research because of its simplicity. The fitness of an individual is equal to the number of bits set to one in its chromosome. This is a very easy problem for GAs since there is no isolation or deception and the BBs are short. The supply of BBs is no problem either, because a randomly initialized population has on average 50% of the correct BBs.

In this function the order of the BBs is $k = 1$. The fitness difference is $d = 1$ because a correct partition contributes one to the fitness and an incorrect partition contributes nothing. The variance may be calculated as $\sigma_{bb}^2 = (1-0)/4 = 0.25$ (see the paper by Goldberg, Deb, and Clark (1992) for a discussion on how to approximate or bound σ_{bb}^2). The first set of experiments uses strings with $m = 100$ bits. Substituting these values into equation 2.4 gives the probability of choosing correctly between two competing BBs as $p = 0.5565$. Since the length of the BBs is one, crossover cannot disrupt them, and uniform crossover was chosen for this function as it mixes BBs quite effectively. The probability of exchanging each bit was set to 0.5.

In figure 2.4 the bold line is the theoretical prediction of the gambler's ruin model and the dotted line is the experimental results. The thin line in the figure is the theoretical prediction of the population sizing model of Goldberg, Deb, and Clark (1992). The results of a second set of experiments with a 500-bit one-max problem are shown in figure 2.5.

The gambler's ruin model predicts the outcome of the experiments for the 100 and 500-bit functions quite accurately. However, in the 500-bit function the match is

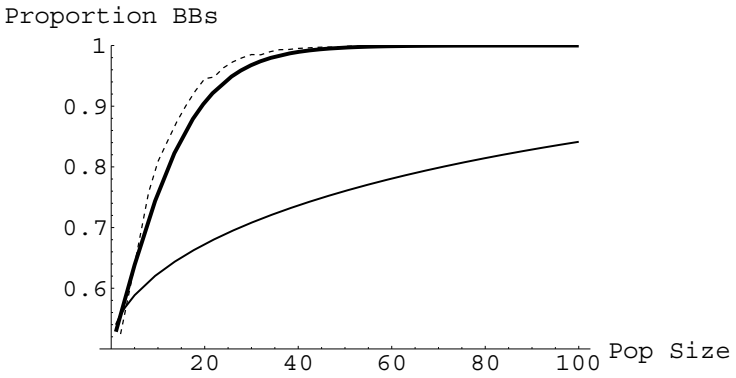


Figure 2.4: Experimental and theoretical results of the proportion of correct BBs on a 100-bit one max function. The prediction of the random walk model is in bold, the experimental results are the dotted line, and the previous decision-based model is the thin line.

not as close as in the 100-bit case. The reason for this small discrepancy may be that the theory only considers one partition at a time and it assumes that decisions for one partition are independent of all the others. In order to achieve this independence, crossover must distribute BBs completely at random across all the individuals in the population. In this case, the goal of crossover is to smooth the distribution of BB in the different alleles and to avoid hitchhiking. However, it would not be practical to reach this perfect distribution, because many rounds of crossover would be necessary in every generation. The problem of inadequate mixing is aggravated as longer strings are used. The predictions of the model should be much more accurate for algorithms such as PBIL (Baluja, 1994), UMDA (Mühlenbein & Paaß, 1996), or the compact GA (Harik, Lobo, & Goldberg, 1998), which treat each bit independently and do not suffer from the inadequate mixing problem described here.

2.4.2 Scaled Problems

The next set of experiments test the predictions of the model on scaled subfunctions. The fitness function is similar to a one-max, except that the contribution of every

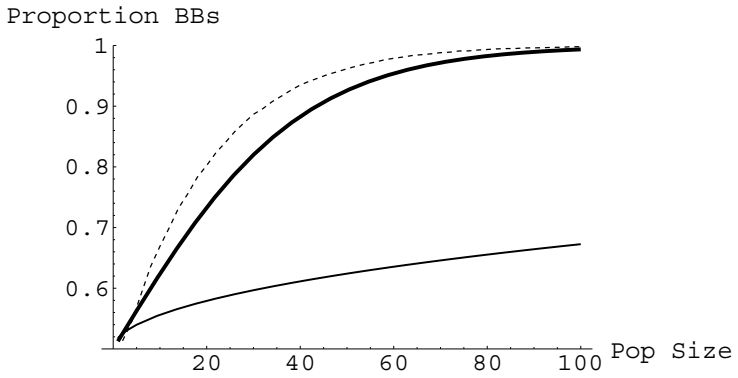


Figure 2.5: Experimental and theoretical results of the proportion of correct BBs on a 500-bit one max function. The bold line is the prediction of the model, the dotted line is the experimental results, and the thin line is the previous decision-only model.

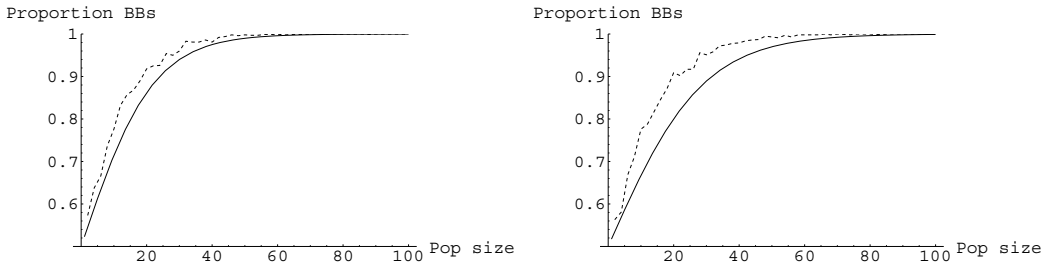
tenth bit to the fitness is reduced. In particular, the first scaled function is

$$F_{sc} = \sum_{i=0}^{m} c_i x_i,$$

where $x_i \in \{0, 1\}$, and $c_i = 0.8$ when $i \in \{0, 10, 20, ..., 90\}$ and $c_i = 1$ otherwise.

The signal of the scaled partitions is smaller than before ($d = 0.8$), and the noise is reduced slightly ($\sigma^2_{bb} = 0.245$). The reduction of the signal makes it harder to decide correctly between the right and wrong BBs, and therefore larger populations are required to reach the correct solution (in this case $p = 0.5457$). The parameters for the experiments are the same as for the one-max functions, but the results reported are the proportion of badly scaled bits that converged correctly. The stronger bits converge more easily, and including them in the results would paint an overly optimistic picture.

The second scaled function is similar to the first, but the signal of every tenth bit is reduced further to $d = 0.6$ (and p becomes 0.5346). The predictions and the empirical results for the two scaled functions are displayed in figure 2.6. In both cases the model correctly predicts larger population sizes, but it is more conservative



(a) $d = 0.8$ (b) $d = 0.6$

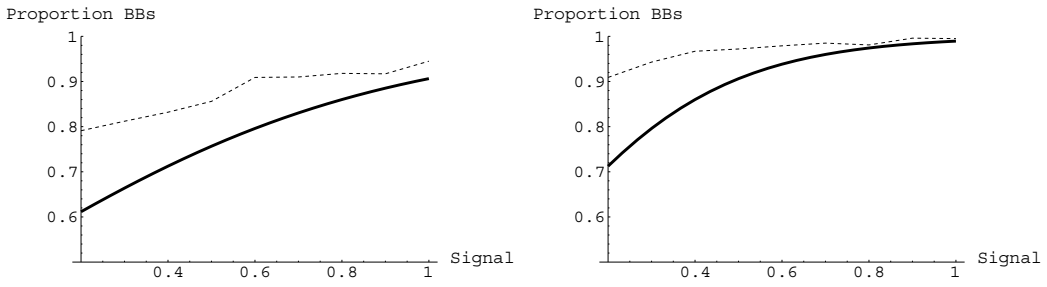
Figure 2.6: Experimental and theoretical results of the proportion of correct BBs on two scaled 100-bit one-max problems. The predictions of the random walk model are plotted in bold and the experimental results with a dotted line.

than with the uniformly-scaled one-max. A probable explanation for this is that the estimate of variance might be too high in the scaled cases. In the GA, the more salient BBs—those that are not scaled down—converge first. Therefore, when the competitions between the scaled BBs begin, the noise coming from the other partitions is much lower because most of them must have converged already.

The effect of reducing the signal even further may be observed in figure 2.7. In this case, the population size is maintained constant while the signal of the scaled bits ranges from 0.2 to 1. The model correctly predicts that the proportion of partitions that converge correctly decreases as the signal becomes weaker. The model is more accurate as the signal increases, but it is always conservative and may be used confidently to determine the population size required to find the desired solution.

2.4.3 Deceptive Functions

The next 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 it can be regulated easily (Deb & Goldberg, 1993). Trap functions are hard for traditional optimizers, because they will tend to climb to the deceptive peak, but



(a) $n = 20$ (b) $n = 40$

Figure 2.7: Experimental and theoretical results on scaled 100-bit one-max functions varying the signal from 0.2 to 1. The predictions are plotted in bold and the experiments with a dotted line. The population size remained constant in each experiment.

GAs with properly sized populations can solve them satisfactorily. We expect to use larger population sizes than before to solve these functions for two reasons: (1) the BBs are much scarcer in the initial population because they are longer; and (2) the signal to noise ratio is lower, making the decision between the best and the second best BBs more difficult.

To solve the trap functions, tight linkage was used (i.e., the bits that define each trap function are positioned next to each other in the chromosome), although there are algorithms such as the messy GA (Goldberg, Korb, & Deb, 1989) and its relatives (Goldberg, Deb, Kargupta, & Harik, 1993; Kargupta, 1996; Harik & Goldberg, 1996) that are able to find tight linkages automatically. Other modern GAs that explicitly identify BBs and treat them independently, such as the extended compact GA (Harik, 1999) and the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1999) can benefit immediately from the research reported in this dissertation.

The first deceptive test function used in the experiments is based on the 4-bit trap function depicted in figure 2.8, which was also used by Goldberg, Deb, and Clark (1992) in their study of population sizing. As in the one-max, the value of this function depends on the number of bits set to one, but in this case the fitness

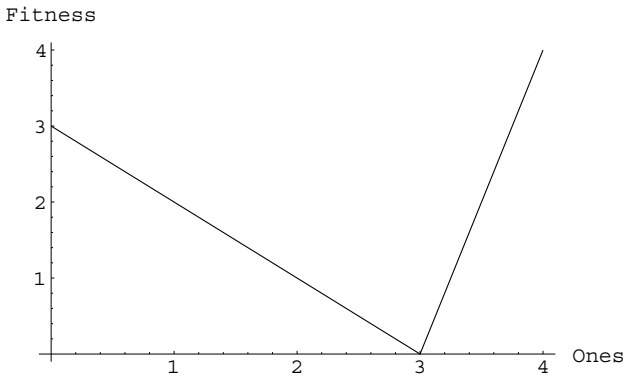


Figure 2.8: A 4-bit deceptive function of unity.

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. The signal difference d (the difference between the global and the deceptive maxima) is 1 and the fitness variance (σ^2_{bb}) is 1.215. The test function is formed by concatenating $m = 20$ copies of the trap function for a total string length of 80 bits. The probability of making the right decision between two individuals with the best and the second best BBs is $p = 0.5585$.

The experiments with this function use two-point crossover to avoid the excessive disruption that uniform crossover would cause on the longer BBs. The crossover probability is set to 1.0, and as before there is no mutation. Figure 2.9 presents the prediction of the percentage of BBs correct at the end of the run along with the results from the experiments. The bold line is the prediction of the random walk model and the thin line is the prediction of the previous decision-only model. Note that the convergence quality for small population sizes is dominated by the initial supply of BBs, and as expected, the previous model is not accurate in this region.

The second deceptive test function is formed by concatenating $m = 10$ copies of an 8-bit fully deceptive trap. The 8-bit trap is similar to the 4-bit used above with a

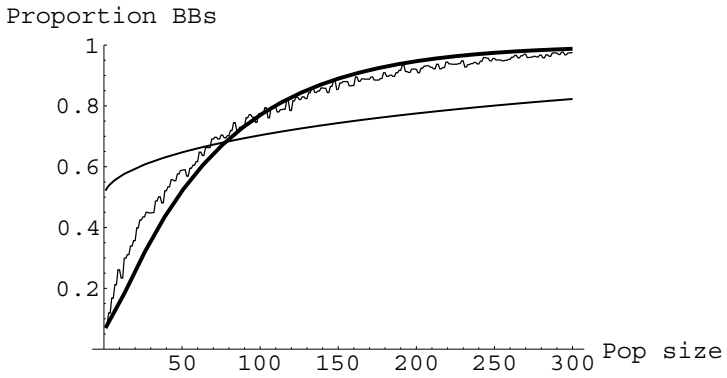


Figure 2.9: Theoretical predictions and and experimental results for a 4-bit deceptive function with 20 BBs. The model (in bold) approximates well the experimental results (dotted). The thin line is the previous decision-making model.

signal difference $d = 1$, but the fitness variance is $\sigma_{bb}^2 = 2.1804$. The higher variance and longer BBs make this function more difficult than the 4-bit problem, so larger populations are expected in this case. One-point crossover was used for this function because the longer BBs are more likely to be disrupted by crossover. The crossover probability was again set to 1.0 and there is no mutation.

Figure 2.10 shows the results for this problem. As in previous cases, the random walk model (in bold) approximates the experimental results well. As expected, the previous population-sizing model (thin line) gives a very conservative estimate of convergence quality, especially for small population sizes.

Experiments reported elsewhere (Harik, Cantú-Paz, Goldberg, & Miller, 1999) suggest that the gambler's ruin model is also accurate on additively-decomposable functions composed of deceptive traps defined over overlapping substrings. These results are encouraging because overlapping subfunctions make the problems more difficult.

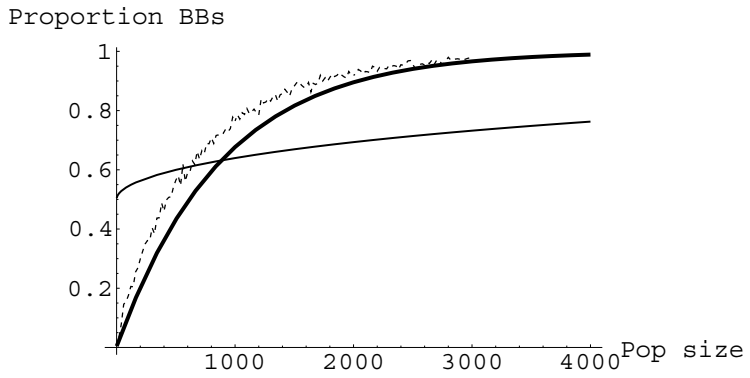


Figure 2.10: Theoretical predictions and experimental results (dotted line) for a 8-bit deceptive function with 10 BBs. The predictions of the gambler's ruin model are in bold, and the previous decision model is plotted with a thin line.

2.5 Noise and Population Sizing

Genetic algorithms are being used increasingly to solve problems where the fitness of an individual cannot be determined exactly due to noise in the evaluation function. The noise may come from an inherently noisy domain or from a noisy approximation of an excessively expensive fitness function. This section examines how this explicit fitness noise affects the size of the population.

Following Goldberg, Deb, and Clark (1992) the noisy fitness F' of an individual is modeled as

$$F' = F + N,$$

where F is the true fitness of an individual and N is the noise present in the evaluation. The effect of the added noise is to increase the fitness variance of the population, making it more difficult to choose correctly between two competing individuals. Therefore, the one-on-one decision-making probability (equation 2.4) has to be modified to include the effect of explicit noise; then, it may be used to find the required population size as was done in section 2.3.

Assuming that the noise is normally distributed as $N(0, \sigma_N^2)$, the fitness variance

becomes $\sigma_F^2 + \sigma_N^2$. Therefore, the probability of choosing correctly between an individual with the optimal building block and an individual with the second best BB in a noisy environment is

$$p = \Phi\left(\frac{d}{\sqrt{\sigma_{H_1}^2 + \sigma_{H_2}^2 + 2\sigma_N^2}}\right).$$

In the case of uniformly scaled problems this becomes (Miller, 1997)

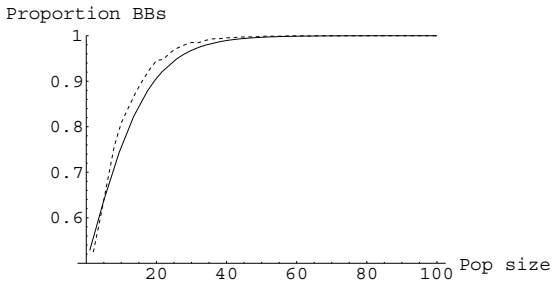
$$p = \Phi\left(\frac{d}{\sqrt{2(m'\sigma_{bb}^2 + \sigma_N^2)}}\right). \tag{2.10}$$

Using this form of p and the same procedure used to obtain equation 2.8 results in the following population sizing equation for noisy domains:

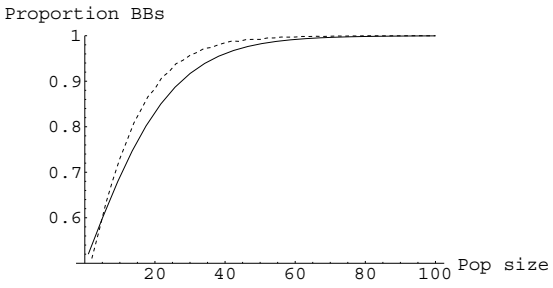
$$n = -2^{k-1}\ln(\alpha)\frac{\sqrt{\pi m'\sigma_{bb}^2 + \sigma_N^2}}{d}. \tag{2.11}$$

The one-max domain was used to test the predictions of the gambler's ruin model (equation 2.5) using the decision probability given by equation 2.10. The experiments used a 100-bit problem with uniform crossover and no mutation. The fitness variance is $\sigma_F^2 = m\sigma_{bb}^2 = 25$ and the experiments were run for σ_N^2 values of $\sigma_F^2 \times (0, 1, 2)$. Figure 2.11 displays the average proportion of partitions that converge correctly (taken over 100 independent runs at each population size) for each noise level. The figure shows that the predictions match well the empirical results.

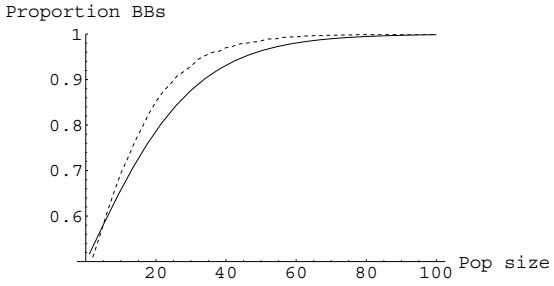
The effect of the increasing noise in the quality of convergence may be observed very clearly in figure 2.12. In this set of experiments, the population size was held constant while the noise increased. The gambler's ruin model correctly predicts that additional noise causes the GA to reach solutions of lower quality, and although the predictions are not exact, the model is conservative and may be used to calculate the required population size.



(a) No noise

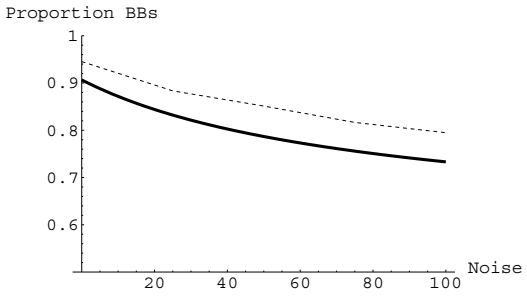


(b) $\sigma_F^2 = 25$

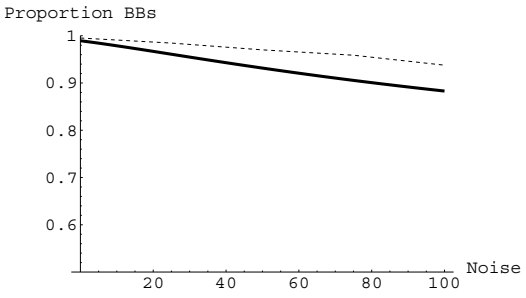


(c) $\sigma_F^2 = 50$

Figure 2.11: Theoretical and experimental results for 100-bit one-max functions with varying noise levels. The noise is constant for each set of experiments, and the population size ranges from 2 to 100.



(a) $n = 20$



(b) $n = 40$

Figure 2.12: Theoretical and experimental results for 100-bit one-max functions varying the noise levels from $\sigma_{bb}^2 = 0$ to 100. The population size remained constant for each set of experiments.

2.6 Effect of Selection Type

Besides the population size, an important factor in the convergence quality of GAs is the selection scheme. After all, the selection mechanism is the part of the GA making the decisions we have discussed in the previous sections. The goal of any selection method is to favor the proliferation of good individuals in the population. However, selection methods differ on how much pressure they exert on the population.

Mühlenbein and Schlierkamp-Voosen (1993) introduced the concept of *selection intensity* to measure the pressure of selection schemes. The selection pressure of tournament selection increases with larger tournament sizes, s. Only tournament selection is considered here, but the results can be extrapolated to other selection methods with known constant selection intensities (see (Bäck, 1994; Miller & Goldberg, 1996)). A different analysis would be necessary for schemes such as proportional selection that do not have a constant selection pressure.

Assume conservatively that the correct BB competes against $s-1$ copies of the second best BB. As larger tournaments are considered, the probability of making the wrong decision increases proportionately to s; thus, we approximate the probability of making the right decision as $1/s$. In reality, this probability is higher than $1/s$ since a tournament might involve more than one copy of the best BB—especially as the run progresses and the proportion of correct BBs increases. However, $1/s$ is a good initial approximation as the experimental results suggest.

The increasing difficulty of decision-making as the tournament size increases can be accounted as a contraction in the signal that the GA is trying to detect. Setting the new signal to

$$d' = d + \Phi^{-1}\left(\frac{1}{s}\right)\sigma_{bb},$$

where $\Phi^{-1}(1/s)$ is the ordinate of a unit normal distribution where the CDF equals $1/s$, we can compute a new probability of deciding well using d' instead of d in

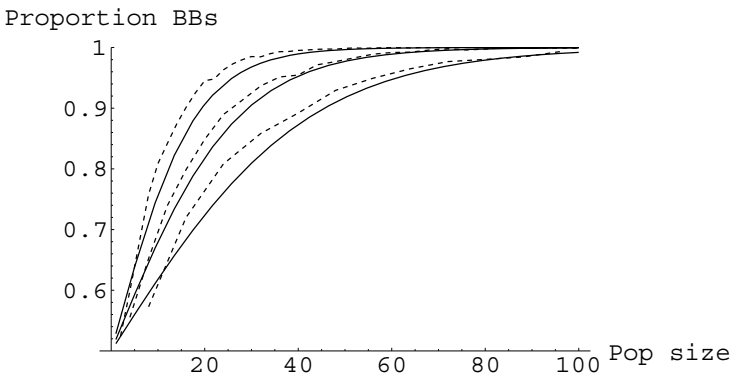


Figure 2.13: Predictions and experimental results for a 100-bit one-max function varying the selection intensity. From left to right: $s = 2, 4, 8$

equation 2.4.

Experiments were performed using a 100-bit one-max function and tournament sizes of 2, 4, and 8. The experimental results are plotted in figure 2.13 with dotted lines along with the theoretical predictions. The leftmost plot corresponds to a tournament size $s = 2$, the next to $s = 4$, and the rightmost to $s = 8$. Once again, the model is a good predictor for the proportion of BBs correct at the end of the run.

2.7 Summary

This chapter presented a solution to a long-standing problem in genetic algorithms: how to determine an adequate population size to reach a solution of a particular quality. The model is based on a random walk where the position of a particle on a bounded one-dimensional space represents the number of copies of the correct BBs in the population. The probability that the particle will be absorbed by the boundaries is well known, and it was used to derive an equation that relates the population size with the required solution quality and several domain-dependent parameters.

The accuracy of the model was verified with experiments using test problems

that ranged from the very simple to the moderately hard. The results confirmed that the model is accurate, and that its predictions scale well with the difficulty of the domain.

The basic model was extended to consider non-uniformly scaled fitness functions, explicit noise in the fitness evaluation, and different selection schemes. Other extensions may require different adjustments, but the facetwise decomposition of tasks that guided this part of the investigation is appropriate.

A correctly sized population is the first step toward competent and efficient genetic algorithms. This chapter showed how to size the population to obtain the desired quality without resorting to unnecessarily large populations, which waste valuable computational resources. The next chapter investigates how to make single-population GAs faster by parallelizing the evaluation of fitness.

Chapter 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 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. However, there has been little analysis to study the benefits that master-slave parallel GAs offer, perhaps because the cost of the frequent communication that is required seems insurmountable.

This chapter examines a very simple master-slave GA. Although faster and more efficient implementations are possible, the objective is to give a simple lower bound on the potential benefits that should be expected of parallel GAs in general.

The execution time of master-slave GAs has two components: the time used in computations and the time used to communicate information among processors. In turn, the computation time is largely determined by the size of the population, so one may be tempted to reduce the population to make the GA faster. However, as the previous chapter discussed, the population size cannot be reduced arbitrarily without suffering a significant decrease in solution quality.

Communication in master-slave GAs occurs every generation when the master sends individuals to the slaves, and when the slaves return the fitness evaluations to

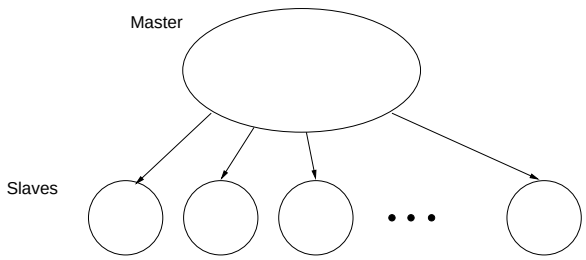


Figure 3.1: A schematic of a master-slave parallel GA.

the master. The cost of communications depends on how many slaves are used and on the hardware platform used to execute the algorithm.

A critical observation in this chapter is there is a tradeoff between increasing communications and decreasing the time that each slave uses to evaluate individuals. This tradeoff is used to find the number of slaves that minimize the execution time of the master-slave GA.

The remainder of the chapter is organized as follows. The next section reviews some of the previous work on master-slave parallel GAs. Section 3.2 finds an accurate predictor of the execution time and shows how to minimize it. Section 3.3 shows experiments with functions of varying computational requirements that validate the accuracy of the theory and show how to calibrate the models to particular hardware and problem domains. Section 3.4 discusses a variation of the simple master-slave algorithm in which the population is distributed between several processors. That section describes the modifications to selection and crossover that are necessary to maintain the illusion of a single panmictic population. This algorithm is important because it similar to a bounding case of multi-deme algorithms that are treated in the remainder of the dissertation. Finally, the chapter ends with a brief summary.

3.1 Background

There are many examples of master-slave GAs in the literature. Most of the publications available are from practitioners who need to find solutions to their applications faster, but there are also a few important theoretical studies.

Bethke (1976) was the first to describe parallel implementations of a conventional GA and of a GA with a generation gap. He performed a detailed analysis of the efficiency of the use of the processing capacity and concluded that it may be close to 100%. However, the analysis ignored the cost of communications.

Another early study of master-slave parallel GAs was made by Grefenstette (1981). He proposed four prototypes for parallel GAs. The first three prototypes were variants of a master-slave scheme, and the fourth was a multiple-deme parallel GA. The first prototype was a synchronous master-slave GA similar to the one examined later in this chapter. In the second prototype, the fitness evaluations and the replacement of individuals occurred asynchronously. The third prototype assumes that the population is stored in a global shared memory, and is also an asynchronous algorithm. In addition, Grefenstette briefly hinted at the possibility of combining multiple populations (the fourth prototype) with any of the three master-slave GAs, to create a hybrid parallel GA. Section 8.1 explores this possibility in more detail.

The early works speculated about the benefits that even simple parallel master-slave GAs would bring to domains with long evaluation times. However, despite the early promises of great speedups, several practitioners who used master-slave GAs stumbled upon a scalability problem: as more processors were used, the efficiency of the parallel algorithm decreased. In most cases, the cause was the increase of communication costs.

For example, Fogarty and Huang (1991) attempted to evolve a set of rules for a pole balancing application that takes a considerable time to simulate. They used a network of transputers, which are microprocessors designed specifically for parallel

computations. A transputer connects directly to only four transputers, and communication between arbitrary nodes is handled by retransmitting messages. This causes an overhead in communications, and in an attempt to minimize it, Fogarty and Huang connected the transputers using different topologies. They concluded that the physical configuration of the network did not make a significant difference in the execution time. Although satisfactory reductions of the execution time were reported, they identified the fast-growing communication overhead as an impediment for further improvements.

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

Hauser and Männer (1994) used three different parallel computers on their experiments with master-slave GAs, but they obtained good results only on a NERV multiprocessor (speedup of 5 using 6 processors), which has a very low communications overhead. They explained the poor performance on the other systems they used (a SparcServer and a KSR1) on the inadequate scheduling of computation threads to processors by the system.

In general, master-slave implementations are more efficient as the evaluations become more expensive. For instance, Grefenstette (1995) obtained about 80% efficiency on a complex robot learning task, but only 50% with an easier task. Moreover, asynchronous implementations that do not have a clear separation between generations are faster than algorithms where the master has to wait for the slower slaves before proceeding (e.g., Zeigler and Kim (1993)).

Besides the evaluation of fitness, another aspect of GAs that can be parallelized is the search operators. For example, recombination and mutation could be parallelized using the same idea of partitioning the population and distributing the work among multiple processors. However, these operators are so simple that any performance gains would be easily offset by the extra time required to send individuals back and forth.

The communications cost also obstructs the parallelization of selection, mainly because several forms of selection need information about the entire population, and would require excessive exchanges of information. In any case, selection does not contribute significantly to the execution time of the GA, and any improvements in selection would have a very small effect overall.

Despite these obstacles, section 3.4 examines an algorithm with a single population that is distributed to several processors. In this algorithm, selection and crossover are parallelized. Although this algorithm requires more communication and is less efficient than the simple master-slave described in the next section, it is important from a theoretical perspective because it resembles the multi-deme algorithms that occupy most of the remainder of the dissertation.

3.2 Analysis

The analysis of master-slave GAs is divided into two parts. The first examines the execution time during one generation of the algorithm. The primary tradeoff between computation and communications is identified, and it is used to find the configuration that minimizes the execution time. The second part of the analysis is concerned with the efficiency of the parallel algorithm and the bound on the possible improvements that should be expected from parallel GAs.

3.2.1 Execution Time

The calculations presented in this chapter make some simplifying assumptions about the components of the master-slave algorithm. First, the time consumed by selection, crossover, and mutation is ignored, because we may assume that it is much shorter than the time used to evaluate and to communicate individuals. In addition, the analysis considers that the number of individuals assigned to any processor is constant, and that the evaluation time is the same for all individuals.

The analysis of the execution time centers on the master processor. Figure 3.2 depicts the sequence of events in every 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, it 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. Implementations where the master stays idle waiting for the results from the slaves are possible, but they would be slower than when the master evaluates a portion of the population.

The slaves start evaluating 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 the evaluation of their shares 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 elapsed time for one generation of the parallel GA may be estimated as

$$T_p = \mathcal{P}T_c + \frac{nT_f}{\mathcal{P}}. \tag{3.1}$$

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

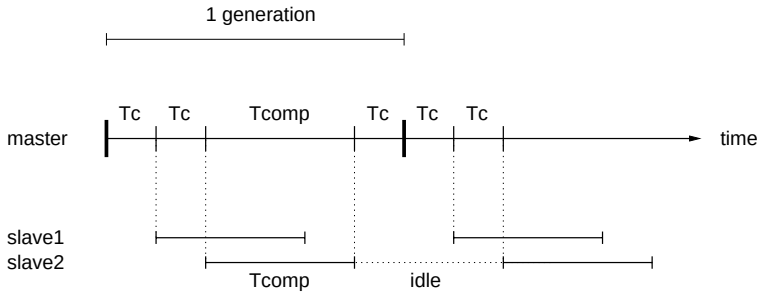


Figure 3.2: A schematic of the execution of a master-slave parallel GA when the master evaluates a fraction of the population.

$\frac{\partial T_p}{\partial \mathcal{P}} = 0$ and solve for $\mathcal{P}$ to obtain

$$\mathcal{P}^* = \sqrt{\frac{nT_f}{T_c}}, \tag{3.2}$$

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

Until this point, the calculations have assumed that T_c is constant. More often, the cost of exchanging information between two processors depends on the amount of information, x. Therefore, a better expression for T_c may be $T_c = Bx + L$, where B is the inverse of the bandwidth of the network and L is the latency of communications. The latency is the overhead per message that depends on the operating system, the programming environment, and on the particular hardware. Since each slave receives $n/\mathcal{P}$ individuals, the time required to send them to the slaves is $T_{send} = B\frac{nl_i}{\mathcal{P}} + L$, and the time to receive the evaluations back is $T_{rec} = B\frac{nl_f}{\mathcal{P}} + L$, where l_i and l_f are the length of the individuals and of the fitness values, respectively. Estimating the elapsed time per generation as $T_p = \mathcal{S}T_{send} + T_{rec} + \frac{nT_f}{\mathcal{P}}$ and optimizing as before, the optimal number of processors becomes

$$\mathcal{P}^* = \sqrt{\frac{n(T_f + B(l_f - l_i))}{L}},$$

which in most cases is not very different from the value given by equation 3.2, because B is a small number that becomes even smaller as networks get faster. The dominant factor in the communications time is the latency, which is constant. Therefore, equation 3.2 is a good estimator of the optimal number of slaves and can be used confidently to design optimal master-slave GAs.

3.2.2 Speedups and the Bound of Acceptable Parallelism

An important concern when implementing master-slave parallel GAs is that the frequent communications 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 is faster than a simple GA the following relationship must hold:

$$\frac{T_s}{T_p} = \frac{nT_f}{\frac{nT_f}{\mathcal{P}} + \mathcal{P}T_c} = \frac{n\gamma}{\frac{n\gamma}{\mathcal{P}} + \mathcal{P}} > 1. \tag{3.3}$$

This ratio is the parallel speedup of the master-slave parallel GA. Solving for γ results in the following necessary condition for better performance in the parallel case:

$$\gamma > \frac{\mathcal{P}^2}{n(\mathcal{P} - 1)}.$$

Substituting the minimum number of processors that may be used in parallel ($\mathcal{P} = 2$) into the equation above results in a very compact condition:

$$\gamma > \frac{4}{n}. \tag{3.4}$$

It is easy to verify if the condition above is true by measuring T_f and T_c on the particular computer that might be used to implement the parallel GA.

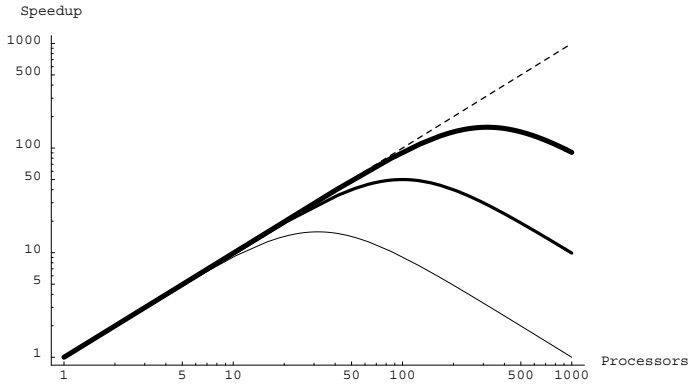


Figure 3.3: Theoretical speedups of a master-slave GA varying the value of γ. The thinest line corresponds to $\gamma = 1$, the intermediate to $\gamma = 10$, and the thickest to $\gamma = 100$. The dotted line is the ideal (linear) speedup.

The inequality above formalizes the intuitive notion that master-slave GAs do not benefit problems with very short evaluation times. The frequent communications would easily offset the reduction in the time required to evaluate the population. However, in most problems of interest the function evaluation time is much greater than the time of communications, $T_f \gg T_c$ ($\gamma \gg 1$), and the population size required to reach an acceptable solution is very large. Therefore, near-linear speedups are possible in many practical problems.

Figure 3.3 shows the theoretical speedups of a master-slave GA varying the value of γ. As γ increases, the range where the speedup is almost linear is wider, and more processors may be used effectively to reduce the execution time. The example considers a master-slave GA with a population of 1000 individuals, and the speedups are plotted for $\gamma = 1, 10, 100$. In practical problems, the values of γ and n may be higher than those used in the example. The ideal speedup is also plotted as a reference.

Another concern about implementing parallel algorithms is to keep the processor utilization high. After all, an algorithm that does not utilize the processing power in the most efficient way might not be the fastest parallel algorithm. However, an

algorithm that utilizes the processors at their maximal capacity is not necessarily the fastest algorithm either (because it may be performing unnecessary computations). In any case, the efficiency of a parallel program is a convenient measure that may facilitate further analysis, and may serve to discriminate against algorithms that waste too many resources.

Formally, the efficiency is defined as the ratio of the parallel speedup divided by the number of processors:

$$E_f = \frac{T_s}{T_p \mathcal{P}}. \tag{3.5}$$

In an ideal situation, the efficiency would always be one and the parallel speedup would be equal to the number of processors used. In reality, the cost of communications prevents this from happening, and the efficiency measures the deviation from the ideal situation. For example, an efficiency of 0.8 means that the algorithm is performing at 80% of its ideal potential. Later in this section we shall see that there is no sense in using an algorithm with an efficiency of less than 0.5.

Figure 3.4 shows an example that depicts how the efficiency drops as more processors are used. The example considers a population of 1000 individuals and $\gamma = 10$. The ideal speedup and the real speedups (as predicted by equation 3.3) are also displayed. In this example, the relatively high costs of communications cause the efficiency to decrease fast, but in situations with a higher γ, the efficiency would remain close to one for a wider range of processors (and the speedup would be near-linear).

We may calculate the critical number of processors that maintain a desired efficiency $\widehat{E_f}$ by making equation 3.5 equal to $\widehat{E_f}$ and solving for $\mathcal{P}$:

$$\mathcal{P}_c = \sqrt{\frac{1 - \widehat{E_f}}{\widehat{E_f}}} \sqrt{\frac{nT_f}{T_c}}, \tag{3.6}$$

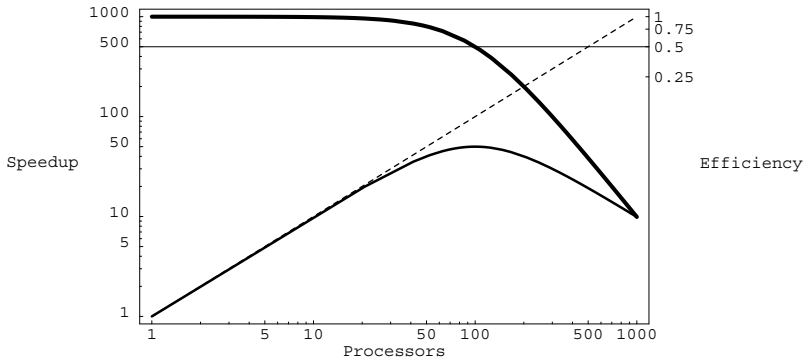


Figure 3.4: The algorithmic efficiency (bold line) decreases as more slaves are used. The figure also shows the ideal speedup (dotted line) and the speedup with communications.

which is a fraction of the optimal number of processors $\mathcal{P}^*$.

An interesting application of this equation is to calculate the critical number of processors where the efficiency equals $1 - \frac{1}{\mathcal{P}}$. At this point, the deviation from the ideal efficiency represents losing one processor to the communications overhead.

To determine this critical number of processors, which we denote as $\mathcal{P}_l$, we substitute $\widehat{E_f} = 1 - \frac{1}{\mathcal{P}_l}$ into equation 3.6. Simplifying terms yields

$$\mathcal{P}_l^3 + \mathcal{P}_l^2 = \frac{nT_f}{T_c} = (\mathcal{P}^*)^2.$$

Asymptotically, we may ignore the $\mathcal{P}_l^2$ term and solve for $\mathcal{P}_l$ to obtain

$$\mathcal{P}_l = (\mathcal{P}^*)^{2/3} = (\gamma n)^{1/3}. \tag{3.7}$$

Besides its benefit as a guide to maintain a high level of processor utilization, the calculation of the critical number of processors (equation 3.6) permits to calculate the maximum speedup of master-slave GAs in a simple way. First, realize that $\mathcal{P}_c = \mathcal{P}^*$ only when $\sqrt{\frac{1-\widehat{E_f}}{\widehat{E_f}}} = 1$, and this may only occur when $\widehat{E_f}$ is 0.5 (see figure 3.4). If the efficiency of the algorithm is 0.5 when the optimal number of processors is used, then using the definition of efficiency

$$E_f = 0.5 = \frac{S_p}{\mathcal{P}^*},$$

and solving for S_p gives the maximum speedup possible as $S_p^* = 0.5\mathcal{P}^*$ or

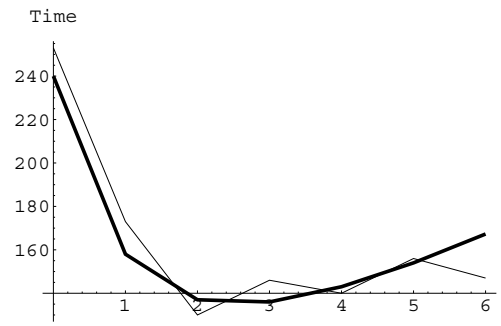
$$S_p^* = 0.5\sqrt{n\gamma}. \tag{3.8}$$

This simple calculation represents a lower bound on the potential speedups of parallel GAs. It is a bound in the sense that other parallel GAs *should* do no worse than the simplest master-slave. While it is possible that other parallel GAs cannot deliver higher speedups, those algorithms should be discarded.

However, there is plenty of room for improvement. At their optimal configuration, master-slave GAs have an efficiency of 50%: half of the time the processors are idle or are communicating with their peers. Surely, there are other parallel GAs that can use the available processing power in more efficient ways and yield solutions of the same quality in less time. The following chapters explore some of those algorithms in detail. The next section presents experimental results that validate the accuracy of the theory for master-slaves.

3.3 Experiments

This section describes experiments with a particular master-slave implementation on a network of IBM RS 6000 workstations. The workstations are connected by a 10 Mbits/sec Ethernet and all communications are implemented using PVM 3.3. This is a rather slow communications environment, and no performance improvements are expected when simple test functions are used. For this reason, the first three sets of experiments use an artificial function that can be altered easily to change its evaluation time (T_f). The fourth set of experiments uses a complex neural network application that takes a very long time to evaluate.



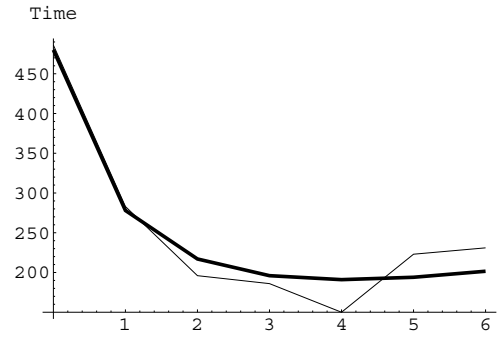
Slaves	Theory	Exp.
0	240	253
1	158	173
2	137	**130**
3	**136**	146
4	143	140
5	154	156
6	147	167

Figure 3.5: Elapsed time (ms) per generation for the 2 milliseconds problem. The thick line on the plot is the theoretical predictions and the thin line is the experimental results. On the table, the numbers with the bold typeface are the minimum execution times.

The artificial test problem for the first three sets of experiments is a dummy function that consists in a simple 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 latency of communications on our system was approximately $T_c = 19$ milliseconds.

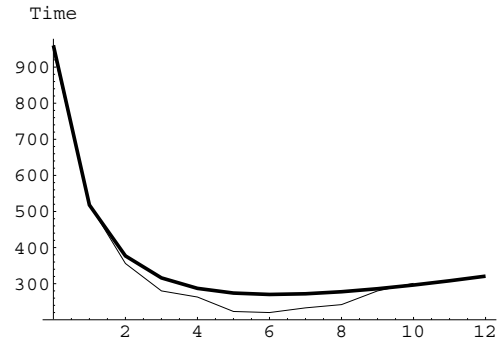
For the first experiment the evaluation time of the test function was set to 2 milliseconds. Using the results of the previous section the optimal number of slaves may be calculated as as $\mathcal{S}^* = \sqrt{\frac{nT_f}{T_c}} - 1 = \sqrt{\frac{120(2)}{19}} - 1 = 2.55$. Figure 3.5 shows the elapsed time per generation of the master-slave parallel GA along with the theoretical prediction (using equation 3.1). The first row of the table is the elapsed time that a serial GA takes to evaluate the population. Note that the master-slave algorithm was faster than the serial GA only when three slaves are used.

In the second experiment, the evaluation time of the test function is doubled to 4 milliseconds, and the results are in figure 3.6. For this case the model predicts that the optimal number of slaves is $\mathcal{S}^* = 4.02$. The evaluation time is doubled again for



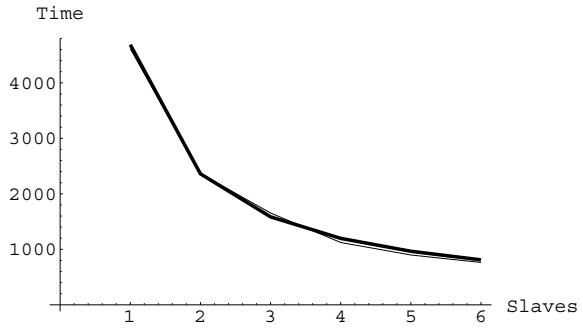
Slaves	Theory	Exp.
0	480	486
1	278	283
2	217	196
3	196	186
4	**191**	**150**
5	194	223
6	201	231

Figure 3.6: Elapsed time (ms) per generation for the 4 milliseconds problem. Theoretical predictions are plotted in bold, and experiments are plotted with a thin line. The bold numbers in the table correspond to the minimum execution times.



Slaves	Theory	Exp.
0	960	946
1	518	520
2	377	356
3	316	280
4	287	263
5	274	223
6	270	220

Figure 3.7: Elapsed time (ms) per generation for the 8 milliseconds problem. Theoretical predictions are plotted in bold, and experiments are plotted with a thin line. The bold numbers correspond to the minimum execution times.



Slaves	Theory	Exp.
0	462	461
1	231	236
2	154	165
3	115	112
4	92	89
5	77	76
6	66	65

Figure 3.8: Elapsed times (sec) per generation for the NN problem.

the third test ($T_f = 8$ ms), and the results of the predictions and experiments are in figure 3.7. In this case $\mathcal{S}^* = 6.1$. Experiments were only conducted for nine slaves because only ten computers were available.

The last experiment uses a more complex evaluation function. It is an application where the GA searched for the weights of the connections of a neural network with 13 inputs, 30 units in the hidden layer, and 5 output units. The objective was to classify a set of 738 vectors that represent sleep patterns into 5 classes. The evaluation function decoded the weights from a string of 5150 bytes, tested each of the patterns, and calculated the percentage of the classifications made correctly. The population size was set to 120 individuals and on our computers each evaluation takes 3.85 seconds, which makes this problem an ideal candidate for the master-slave method. As we can see in figure 3.8 the total elapsed time decreased linearly with the number of slaves because the time used in communications was only a small fraction of the total time. In fact, the communications overhead is so small that for this problem $\mathcal{S}^* = 151$ processors.

3.4 A Distributed Panmictic Population

An alternative to store the entire population in one master node is to distribute the population to several processors. The algorithm has to be modified so that the distributed population behaves as a single panmictic unit. 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 calculations in this section are very similar to section 3.2. 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.

Let n denote the population size required to reach the desired quality in a particular problem. 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 evaluation 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 & 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 on 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 computation time per node is simply $\frac{nT_f}{\mathcal{P}}$. Communications are used during both selection (T_s) and crossover (T_x). The general form of the time used in communications 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 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 communications as $(\kappa_s+1)(\mathcal{P}-1)T_c$. To simplify the calculations we define $\kappa = \kappa_s + 1$. Adding computation and communications times we obtain the total execution time per generation:

$$T_p = \frac{nT_f}{\mathcal{P}} + \kappa(\mathcal{P}-1)T_c. \tag{3.9}$$

As before, when more processors are used there is a tradeoff between decreasing computations and increasing communications. Solving $\frac{\partial T_p}{\partial \mathcal{P}} = 0$ for $\mathcal{P}$ we obtain the optimal number of processors that minimize the execution time:

$$\mathcal{P}^* = \sqrt{\frac{nT_f}{\kappa T_c}}. \tag{3.10}$$

In the case of a traditional master-slave algorithm there is only one communications event per generation ($\kappa = 1$), and the result above agrees with the calculations in section 3.2.

Although there are multiple communications per generation, in most practical situations the function evaluation time is much greater than the time of communications, $T_f \gg T_c$, 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.

3.5 Summary

Master-slave parallel GAs are easy to implement, and all the theory on simple GAs can be used to choose adequate values for the search parameters. However, master-

slave GAs are sometimes ignored because the cost of communications seems insurmountable. The analysis of this chapter showed that for many applications the reduction in computation time is sufficient to overcome the communications cost. Also, the calculations presented here should be useful to design fast master-slave GAs that utilize the computing resources in the best possible way.

The main contribution of this chapter is to present a lower bound on the performance gains that are acceptable in any parallel GAs. More sophisticated single-population GAs or multi-population parallel GAs should do better than the simple case examined here or they should be abandoned. The calculations are simple and they are easy to calibrate to consider the hardware and the particular domain.

The chapter also presented a GA with a single distributed population. The algorithm is less efficient than the simple master-slave, but it is important because it resembles a GA with multiple communicating populations. The analysis of this algorithm is straightforward, and reveals that the optimal number of processors is of the same order as the master-slave.

Chapter 4

Bounding Cases of Multi-Deme GAs

The first part of the dissertation discussed the design of single-population GAs. The second part investigates multi-deme parallel GAs and begins with this chapter. Multi-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 that are relatively isolated from each other, as in islands. Both in the island model and in the GAs, individuals may migrate occasionally to any other population.

As we saw in the introduction, the design of multiple-deme parallel GAs involves difficult and interrelated choices. The three main issues are to determine (1) the size and the number of demes, (2) the topology that interconnects the demes, and (3) the migration rate that controls how many individuals migrate. These three dimensions of the problem are a natural decomposition that will be used in the remainder of this thesis. Additionally, one has to decide how to choose migrants and how to replace existing individuals in a deme, but these issues do not arise in the bounding cases considered in this chapter.

Most studies of parallel GAs are empirical investigations that concentrate on the choices of migration rates and topologies, and treat the population-sizing issue as a secondary problem. However, since the population size largely determines the quality of the solution (Goldberg, Deb, & Clark, 1992; Harik, Cantú-Paz, Goldberg, & Miller, 1997) and the duration of the run (Goldberg & Deb, 1991), it is only natural that

our study of multiple-deme parallel GAs begins by addressing this issue.

However, we cannot ignore the importance of topologies and migration rates, because they have unknown effects on the efficiency and on the quality of the solutions reached by the parallel GAs. 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 these parameters. 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. The model for the completely isolated case is a straightforward extension of the gambler's ruin model introduced in chapter 2. 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 the migration rate to intermediate values, the bounding cases serve as indicators of the performance of parallel GAs. Once the simplifications and assumptions used in this chapter are shown to be viable, the models will be specialized later.

The next section is a brief review of related work on multi-deme parallel GAs. Section 4.2 extends the gambler's ruin model to calculate the deme size for the isolated case and section 4.3 considers fully connected demes. Experiments are presented alongside the theory to validate its accuracy.

4.1 Background

Despite the difficulties in their design, multi-deme parallel GAs have been very popular, and the literature contains many reports of successful implementations. However, there are only a handful of studies that characterize the quality and efficiency of this class of algorithms; further examples can be found elsewhere (Cantú-Paz, 1998).

Grefenstette (1981) was probably the first to describe a multiple-deme parallel GA. In his algorithm, the best individuals from each deme are broadcast every gen-

eration to all the others. The complexity of coarse-grained parallel GAs was evident from this early proposal and Grefenstette raised several "interesting questions" about the frequency of migration, the destination of the migrants (topology), and the effect of migration on preventing premature convergence.

Grosso (1985) was the first to observe some of the phenomena that continue to appear in the literature. He found that the rate of fitness improvement was faster in a population divided into demes than in a single panmictic population. But when the demes are isolated, the final solution's quality was lower than that reached by the single large population. Grosso experimented with a wide range of migration rates, and observed that with intermediate settings the partitioned and the panmictic populations reached solutions with the same quality. In his experiments, migrations occurred every generation, and the migrants and their destinations were selected randomly. Grosso also tried a "delayed" migration scheme in which communications began only after the demes were near convergence. At that point, the demes exchanged individuals with a high migration rate. The delayed scheme reduced the cost of communications and resulted in solutions of the same quality as those obtained with frequent migrations. Later, this chapter presents an analysis of the solution quality that can be reached by an algorithm with a similar migration scheme.

Braun (1990) used a similar idea and presented an algorithm where migration occurred after the demes converged completely (the author used the term "degenerate") with the purpose of restoring diversity into the demes to prevent premature convergence to a low-quality solution. Later, Munetomo, Takai, and Sato (1993) also adopted a similar migration strategy.

Tanese initiated the systematic study of migration and its effects on the efficiency and quality of parallel GAs (Tanese, 1987; 1989a; 1989b). Her experimental method consisted in dividing a fixed number of individuals into equally-sized demes, and to vary the migration rates and the migration intervals.

Tanese's experiments showed that medium migration rates are necessary to find

solutions of the same quality as with a serial GA. A migration rate too low or too high would result in lesser solutions. Like Grosso, Tanese also examined completely isolated demes and found that the solutions were generally inferior to those found by a serial GA or by the parallel GA with migration. In terms of the efficiency of the parallel algorithm, she found that "near-linear" speedups can be obtained when migration is infrequent and the migration rate is low.

Implicit in Tanese's experimental method is the assumption that several small populations can be compared fairly against one aggregate population. Keeping the total number of individuals constant seems to be an adequate premise for a fair experimental design, but since the quality and cost depend non-linearly on the deme size, neither the cost or the quality were held constant in the experiments. Unfortunately, others also treat the questions of efficiency and quality separately, and—not surprisingly—reports of linear, quasi-linear, or super-linear speedups appear constantly in the literature. For example, Belding (1995) recently confirmed Tanese's findings on the effects of migration on quality, and he showed superlinear speedups for up to 32 processors. The controversial claims of superlinearity in parallel GAs will be considered further in the next chapter.

Nevertheless, Tanese's groundbreaking study identified many of the important issues for the correct design of parallel GAs: the choice of one or many populations, the frequency of migrations, and the migration rate. In addition, she studied how the migration parameters affect the loss of allele diversity. To prevent the premature loss of useful solutions, she proposed to set the GA parameters differently in each population, forcing some populations to converge slower than others. Diversity is preserved longer in the slowly converging populations, and migration would reintroduce diversity into the populations that were set to converge fast. Later, others would adopt this idea (e.g., (Lin, Punch, & Goodman, 1994)).

This chapter uses an approach that differs from most investigations of parallel GAs. It acknowledges that the problems of solution quality and algorithmic efficiency

are strongly related, and the analysis considers them simultaneously. This is the main reason why some of the results of this chapter and the next may disagree with the results of others.

4.2 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.

4.2.1 Expected Quality and Deme Size

The first step in the 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 that converge correctly (X), and under the assumption that partitions are independent from each other, the quality has a binomial distribution with parameters m and P_{bb}. We may write the qualities of the solutions of the r demes in ascending order as

$$X_1 \leq X_2 \leq ... \leq X_r.$$

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 then five, but these values have been tabulated extensively for the standard normal distribution

(see table 4.1, taken from (Harter, 1970)). 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

$$\hat{Q} = E(X_r) = mP_{bb} + \mu_{r:r}\sqrt{mP_{bb}(1 - P_{bb})}. \tag{4.1}$$

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 as 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

$$\hat{Q} = E(X_r) \leq mP_{bb} + \frac{\mu_{r:r}}{2}\sqrt{m}. \tag{4.2}$$

We ignore the inequality and solve this equation for P_{bb} to obtain the required probability of success per deme as

$$\hat{P} = \frac{\hat{Q}}{m} - \frac{\mu_{r:r}}{2\sqrt{m}}. \tag{4.3}$$

This equation clearly shows how the required probability of success per deme is relaxed as more demes are used. It may be approximated asymptotically using $\mu_{r:r} \approx \sqrt{2 \ln r}$ (Beyer, 1993) as[1]

[1] I discovered, almost accidentally, that $\mu_{r:r}$ can be approximated more accurately as $\mu_{r:r} = \sqrt{\sqrt{2} \ln r}$. However, this does not change the conclusions of this section.

r	1	2	4	8	16	32	64	128	256
$\mu_{r:r}$	0	0.5642	1.029	1.423	1.766	2.069	2.343	2.594	2.826

Table 4.1: 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.

$$\hat{P} = \frac{\hat{Q}}{m} - \frac{\sqrt{\ln r}}{\sqrt{2m}},$$

which shows that $\hat{P}$ decreases very slowly with respect to r.

To find the deme size we use the gambler's ruin problem in a manner similar to section 2.3. 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

$$n_d = \frac{2^k \ln(1-\hat{P})}{\ln\left(\frac{q}{p}\right)}, \tag{4.4}$$

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

$$n_d = 2^{k-1} \ln(1-\hat{P}) \sqrt{\pi m'} \frac{\sigma_{bb}}{d}. \tag{4.5}$$

This equation is similar to the population size of the simple GA found in section 2.3. 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 $r = 1$).

4.2.2 An Different Derivation

Cantú-Paz and Goldberg (1997) used a different formulation to reach the same result of the previous section. Here we will show that their approach is equivalent to the one presented in the previous section.

The premise of their method is that the probability that one deme reaches the desired solution increases as more demes are used. Define P_m as the probability that

one deme finds the desired solution $\hat{Q}$, then $(1-P_m)^r$ is the probability that none of the r demes succeeds. Thus, the probability that at least one of the demes reaches the solution is

$$P_s = 1-(1-P_m)^r. \tag{4.6}$$

Let $\mathrm{B}(x)$ denote the CDF of a binomial distribution with parameters m and P_{bb}. The probability that a deme converges to a solution with *at least* $\hat{Q}$ out of m partitions correct may be calculated as $P_m = 1-\mathrm{B}(\hat{Q})$. Substituting this form of P_m into equation 4.6, and simplifying results in

$$P_s = 1-\mathrm{B}^r(\hat{Q}) \tag{4.7}$$

Section 2.3 showed how to compute the population size needed by the simple GA to reach a solution with an average of $\hat{Q}$ out of m BBs correct. An equivalent result may be found by solving $P_m = 0.5$ for n, because the binomial distribution is symmetrical around the mean. To be fair and compare the two algorithms at the same success probability, the parallel deme size has to be found solving $P_s = 0.5$ for n. Making $P_s = 1-\mathrm{B}^r(\hat{Q}) = 0.5$ and simplifying gives

$$P_s = \mathrm{B}^r(\hat{Q}) = 0.5.$$

The left hand side of the equation is the CDF of the highest-order statistic of a binomial distribution. The problem is to determine the point where this CDF reaches 0.5. In other words, the problem is to determine the mean of the highest-order statistic, which is the same problem solved in the previous section.

Before examining the other bounding case where the demes communicate with all others, the next section presents empirical evidence that validates the results for

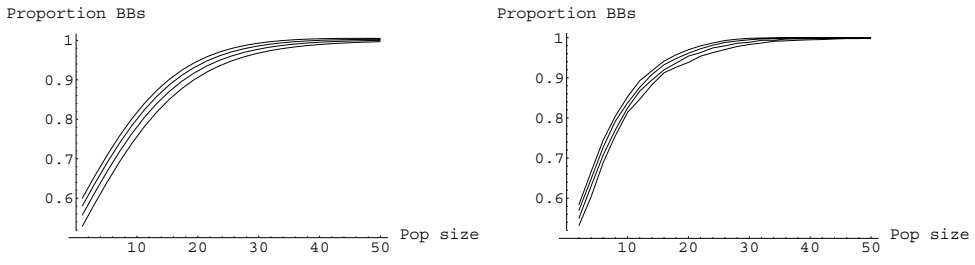


Figure 4.1: Theoretical predictions and experimental results for the proportion of partitions correct in the *best* isolated deme. The graph shows results using 1,2,4,and 8 demes (from right to left), and considers a one-max function with 100 BBs.

the case with isolated demes.

4.2.3 Experiments

The experiments in this chapter use a simple GA with pairwise tournament selection and a crossover operator without excessive disruption. As such, the crossover operator was domain-dependent, and the choice is specified as each problem is presented. In all cases, the crossover probability is set to 1.0, and the mutation probability is zero. The results presented are the average over 100 independent runs. The experiments were performed on a simulated parallel environment consisting of eight IBM RS6000 workstations connected by a 10 Mbps Ethernet. A virtual parallel computer was simulated using PVM 3.3.

The first test problem is a one-max function with $m = 100$ bits. Uniform crossover is used because it provides good mixing of BBs, and BB disruption is not an issue because the BBs have length equal to one. Figure 4.1 compares the theoretical predictions for the proportion of BBs with experimental results using 1, 2, 4, and 8 demes.

The second test function is based on the 4-bit trap function described in chapter 2. Since the BBs in this function are longer than in the one-max problem, a two-point crossover operator was chosen to avoid the excessive disruption of uniform

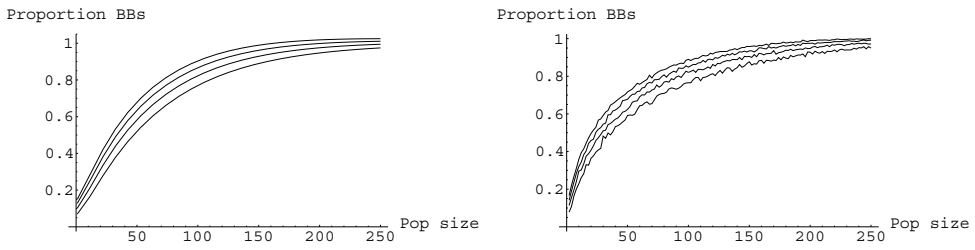


Figure 4.2: Theoretical predictions and experimental results for the proportion of partitions correct in the *best* isolated deme. The graph shows results using 1,2,4,and 8 demes (from right to left), and considers a 4-bit trap function with 20 BBs.

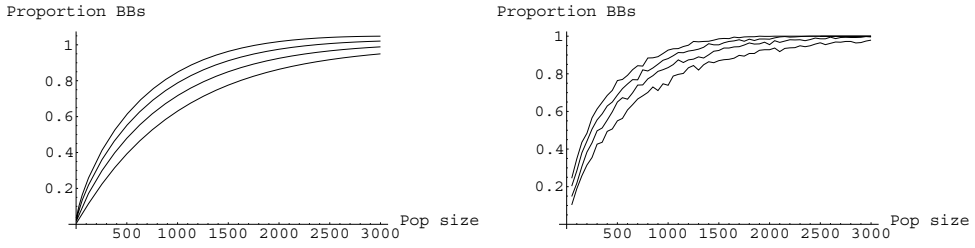


Figure 4.3: Theoretical predictions and experimental results for the proportion of partitions correct in the *best* isolated deme. The graph shows results using 1,2,4,and 8 demes (from right to left), and considers a 8-bit trap function with 10 BBs.

crossover. Figure 4.2 presents the prediction of the proportion of partitions correct along with the experimental results. The third set of experiments used an 8-bit trap function, and the GAs were configured to use one-point crossover. The predictions and empirical results are displayed in figure 4.3.

In all the experiments the model predicts accurately a small increase in quality as more demes are used.

4.3 Fully Connected Demes

Executing demes in complete isolation is clearly a bound for parallel GAs, as 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 other important free parameter is the migration frequency, and following the methodology of this dissertation we will study the two bounding cases for this parameter.

The upper bound of the frequency of migration is to exchange individuals every generation. This extreme case represents the most intense communication possible: fully connected demes, maximal migration rates and communication occurs every generation. Notice that the algorithm with a distributed panmictic population described in section 3.4 resembles this bounding case; in that algorithm each processor executes essentially a complete GA (fitness evaluation, selection, crossover, and mutation), and there is an exchange of individuals every generation. The topology of communication is a fully connected graph, because each processor exchanges individuals with every other during selection and crossover, and the number of individuals exchanged is $n/\mathcal{P}$, which corresponds to the maximal migration rate. Section 3.4 presented how to estimate and minimize the execution time of this algorithm.

However, the distributed population algorithm has *multiple* communications per generation and this is its major difference with multi-deme GAs, which have *at most one* migration event in a generation. In multi-deme 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. Apparently there have been no studies of this bias, but it may be not very strong 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, except when both the migrants and the individuals they replace are chosen randomly (Cantú-Paz, 1999a; 1999b), which was the case used in the distributed panmictic algorithm. If a different migration policy is used, the multi-deme algorithm may converge faster.

The differences between the distributed panmictic population and fully connected demes that communicate every generation with the maximal migration rate are very small, and we can expect that they reach solutions of the same quality (which can be predicted with the gambler's ruin model). If this is the case, then the calculations in section 3.4 are appropriate for this bounding case.

The remainder of the dissertation considers multi-deme GAs that use the *lower* bound of the migration frequency. At this bound, migration occurs after the demes have converged (i.e., all the individuals in each deme are identical, not necessarily representing an optimal solution). After convergence, each deme partitions its population into r mutually exclusive sets, and exchanges the individuals of the i-th set with the i-th deme. The demes restart after migration, and execute the GA operations until they converge again. One advantage of this bounding case is that 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.

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; chapters 6 and 7 extend the analysis to multiple epochs.

4.3.1 Success Probability and Population Size

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 present initially. When the demes converge the first time, a partition either has n_d copies of the correct BB with probability P_{bb}, or it has n_d copies of the wrong BB 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 BB, but some have the correct one. Therefore, after migration each partition has on average nP_{bb} copies of the correct BB, which means that the second epoch starts from a better starting point than the first. To calculate the probability that the deme converges correctly this time the gambler's ruin model may be used again; we only need to replace x_0 with the new starting point as follows:

$$P_{bb_2} = 1 - \left(\frac{q}{p}\right)^{n_d P_{bb}}. \tag{4.8}$$

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

$$E(X_r) = mP_{bb_2} + \mu_{r:r}\sqrt{mP_{bb_2}(1 - P_{bb_2})}. \tag{4.9}$$

This equation shows that 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 4.3). The second step is to solve $\hat{P} = P_{bb_2}$ for the deme size, n_d.

We would like to use equation 4.8, 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 4.8 has to be approximated and the deme size will be derived from the approximation. Equation 4.8 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

$$P_{bb_2} \approx 1 - \exp(-cn_d P_{bb}). \tag{4.10}$$

Similarly, P_{bb} can be rewritten exactly as $P_{bb} = 1 - (1-c)^{n/2^k}$ and approximated as $P_{bb} \approx 1 - e^{-cn/2^k}$. 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 4.10 results in

$$P_{bb_2} \approx 1 - \exp\left(\frac{-c^2 n_d^2}{2^k}\right). \tag{4.11}$$

With this form of P_{bb_2}, we can now solve $\hat{P} = P_{bb_2}$ and find a deme-sizing equation

$$n_d = \frac{\sqrt{-2^k \ln(1-\hat{P})}}{1 - \frac{q}{p}}, \tag{4.12}$$

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

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

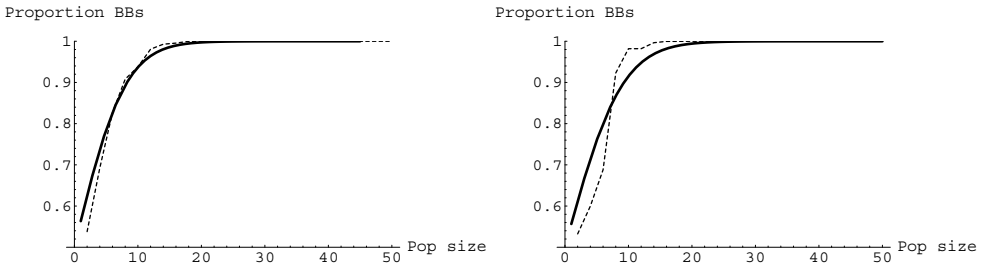
The deme size depends on the *square root* of $2^k \ln(1-\widehat{P_{bb}})$ as opposed to the case for isolated demes where the deme size is directly proportional to this term. Although the approximation is coarse, this equation clearly shows that a parallel GA with migration needs much smaller demes, which in turn represent a substantial reduction of the time the GA uses in computations.

4.3.2 Experiments

As with the isolated demes case, the one-max and two deceptive trap function are used to test the model for the fully connected topology. The experiments use 4 and 8 demes with their corresponding maximal migration rates of 25% and 12.5%.

Figure 4.4 shows the predictions of the average proportion of BBs correct (P_{bb_2}) against the population size for a 100-BB one-max function. As discussed before, the one-max problem is very easy for GAs and small population sizes are sufficient to find solutions of high quality. Also, note that when using 8 demes and a migration rate of only 12.5%, migration is not expected to occur for population sizes smaller than 12 individuals. This is evident on the left portion of figure 4.4 where the theory predicts a better convergence quality of the GA for low population sizes. Once the population size is large enough to make migration possible, the GA performs in better accordance with the model.

The results for the experiments with the trap functions are plotted in figures 4.5 and 4.6. These functions require larger population sizes than the one-max, and thus the gap between the prediction and the experimental results at low population sizes is much smaller. Also, the approximation of the initial starting point of the second iteration becomes better as more demes are used. For this reason, the model predicts



(a) 4 demes, 25% migration

(b) 8 demes, 12.5% migration

Figure 4.4: Theoretical predictions and experimental results for a one-max function with 100 BBs. Theoretical predictions are in bold.

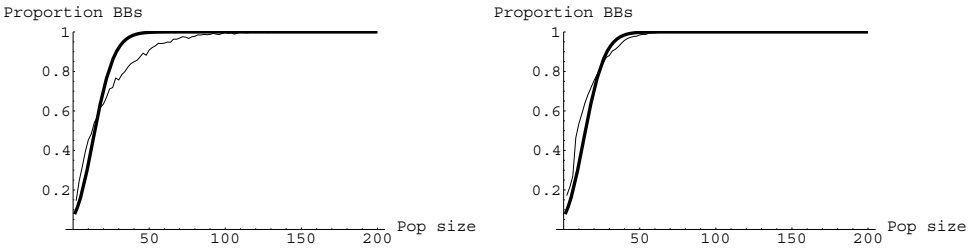
more accurately the performance of the parallel GAs with more demes.

4.4 Summary

The analysis presented in this chapter recognizes that the design of parallel GAs is a complex problem, and that the choices of topologies, migration rates, number of demes, and their size are intimately related. To make progress on the deme sizing problem without ignoring the other choices, the analysis used bounds on the topologies and migration rates.

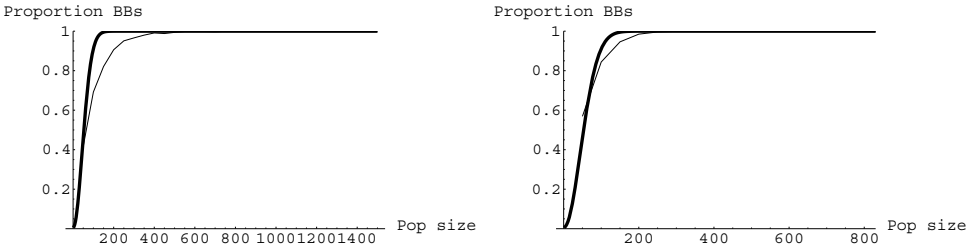
The starting point of the analysis was to calculate the relaxation of the target quality per deme. Then it was possible to compute the expected quality of the best deme for each bounding case, and those results were combined with the gambler's ruin model to derive deme-sizing equations. Finally, the accuracy of the models was verified with experiments.

An important result of the analysis is that the population size decreases very modestly when there is no communication between the demes. But when the demes



(a) 4 demes, 25% migration (b) 8 demes, 12.5% migration

Figure 4.5: Theoretical predictions and experimental results for a 4-bit trap function with 20 BBs. Theoretical predictions are in bold.



(a) 8 demes, 12.5% migration (b) 16 demes, 6.25% migration

Figure 4.6: Theoretical predictions and experimental results for a 8-bit trap function with 10 BBs. Theoretical predictions are in bold.

communicate, the required deme size is much smaller. The next chapter examines how these reductions of the deme sizes translate into cost savings. Chapter 6 will extend the analysis of the fully connected case to consider multiple epochs, and chapter 7 will considered other topologies and reduced migration rates.

Chapter 5

Parallel Speedups of the Bounding Cases

This chapter shows how to calculate the parallel speedups of the two bounding cases introduced in the previous chapter. The results show that the speedup is not very significant when the demes are isolated, but there is a substantial improvement in performance when the demes communicate. Moreover, in the fully connected case there is a combination of number of demes and deme size which together minimize the parallel speedup, and this chapter shows how to calculate it.

The chapter begins with a discussion about the controversial claims of superlinear speedups in parallel GAs. Then, sections 5.2 and 5.3 present the calculations of the speedup of isolated and fully-connected demes, respectively. The chapter ends with a brief summary in section 5.4.

5.1 Parallel Speedups

The general purpose of parallel computation is to reduce the time necessary to reach a solution in a particular problem. To measure the benefit of using a parallel algorithm we may calculate the gain in execution time that results from using multiple processors. This gain is usually normalized with respect to the serial time, and expressed as the parallel speedup of the algorithm.

In the GA community there has been some controversy about parallel speedups. One of the primary points of disagreement are the frequent claims of superlinear speedups (i.e., the execution time of the parallel GA is reduced by a factor greater

than the number of processors used). The remainder of the section addresses these claims, and describes how the speedups are measured in this investigation [1].

In general, to make a fair comparison between any serial and parallel programs the two must give exactly the same result; and in the case of stochastic algorithms like GAs, a fair comparison must be based on the *expected* quality of convergence. As discussed in the previous chapter, some claims of superlinear speedups are suspicious because serial and parallel GAs were compared without explicitly considering the quality of the solutions. Sometimes an experimental design that seems fair because it holds constant one of the important variables (like the total number of individuals in Tanese's study) is inadequate because the solution quality is not guaranteed to be the same. The analysis in this chapter avoids this problem, and makes fair comparisons of serial and parallel GAs by using the models presented in the previous chapter, which predict the deme sizes needed to obtain *on average* a solution of the desired quality.

Another reason to suspect the claims of superlinear GAs is an argument commonly used in parallel processing to dismiss superlinearity in general. The argument is that if all the tasks of a parallel program are executed on a single processor, the total execution time cannot be less than the execution time of a serial program that performs the *same* computations. As Punch (1998) points out, the key point is the assumption that the serial and parallel GAs execute the exact same tasks.

In this view, the only possible explanation of superlinear speedups is that the parallel GA does not execute the same amount of work as the serial GA. Somehow it executes less. Unfortunately, the current explanations of the reduction of work are

[1] We ignore the "trivial" explanations of superlinear speedups that arise from details of particular implementations. In particular, it is possible that small demes fit completely in the processors' caches, reducing the memory access times so much as to produce superlinear speedups. This was an explanation offered by Belding (1995) about his results. Another case that we ignore is the use of the parallel code to obtain timings for the serial case. This may artificially inflate the serial execution time with unnecessary operations.

not satisfactory. While some studies recognize that the choices of migration rates, frequency of migration, choice of migrants and topology have some effect on the amount of work, they fail to ask *why* the work is reduced sometimes.

What causes multiple-deme GAs to do less work than GAs with a single population? This is a very complex question that requires further research, and its complete exploration is outside the scope of this dissertation. Here, we briefly examine two possible reasons.

First, it is very likely that multiple populations evolve to different regions of the search space, and therefore migration may introduce to each deme individuals that are different from the native ones. The increased diversity in each deme may have two effects. One is that the increased diversity will delay the convergence of the algorithm. This may give enough time[2] to the crossover operator to mix BBs together into solutions of higher quality than those reachable by a simple GA with an aggregate population (Goldberg, 1998). This means that the demes may be reduced and still reach the same solution as the serial GA, and, in some cases, the reduction may be sufficient to obtain superlinear speedups. The other effect of introducing diverse individuals to a deme is that it may be possible to evolve partial solutions independently in different demes and integrate them after migration (Whitley, Rana, and Heckendorn (1999) have reported some promising work about this possibility considering separable functions). In this case, it is also possible to reduce the deme sizes because it is not necessary to solve the entire problem at once.

A second possible explanation of the reduction of the work in multi-deme GAs is that the selection pressure is affected by the choice of migrants and by how they replace individuals at the receiving deme (Whitley, 1993b). It has been established that higher selection pressures cause faster convergence (see for example the papers

[2]See the papers by Goldberg, Deb, and Thierens (1993) and Thierens and Goldberg (1993) for a discussion on the time required to mix BBs into good solutions and the relation of this "innovation time" with the success of the search.

by Mühlenbein and Schlierkamp-Voosen (1993) and Bäck (1994)), but until recently the effect of the migration policy on the selection pressure has not been studied (Cantú-Paz, 1999a; 1999b).There are two choices to select the individuals that migrate: choose them randomly or pick the best individuals in the population. Likewise, there are two choices at the receiving deme to replace existing individuals with the incoming migrants: choose randomly or replace the worst. The issue is that when migrants or replacements are chosen according to their fitness, the algorithm's selection pressure increases, and therefore the populations converge in fewer generations.

Each of the four combinations of migrant selection and replacement results in a different selection pressure. The pushiest scheme is to select the best individuals to migrate and replace the worst, because fitness-based selection is performed in both the sending and receiving demes. This is the scheme commonly used in multiple-deme GAs, and it may cause superlinear speedups if the parallel GA is improperly compared against a serial GA with the original—weaker—selection pressure. Cantú-Paz (1999) presents detailed calculations of the selection intensity under different migration policies and an example of an inappropriate comparison of serial and parallel GAs.

The issues of improved mixing and higher selection pressure do not appear in the algorithms examined in this thesis. These two effects of migration can only occur when the demes exchange individuals before they converge, and this investigation considers that migration occurs after convergence.

Besides superlinear speedups, another source of controversy is that often it is not clear how performance is measured. Sometimes it is convenient to use the number of function evaluations, because different algorithms may be compared regardless of their implementation details. More often, the performance is measured by recording the wall-clock time, so all the components of the execution time (especially communications) are accounted. This is a fair measure, and it is the one used in this study.

The analysis assumes that there is one processor for every deme, so the parallel execution time, T_p, is the time used by *one* deme to complete its tasks (computations and communications). Another assumption is that the computation time is dominated by the time used to evaluate the objective function. The GA operations—selection, crossover, and mutation—are relatively simple, and thus the analysis ignores them.

5.2 Isolated Demes

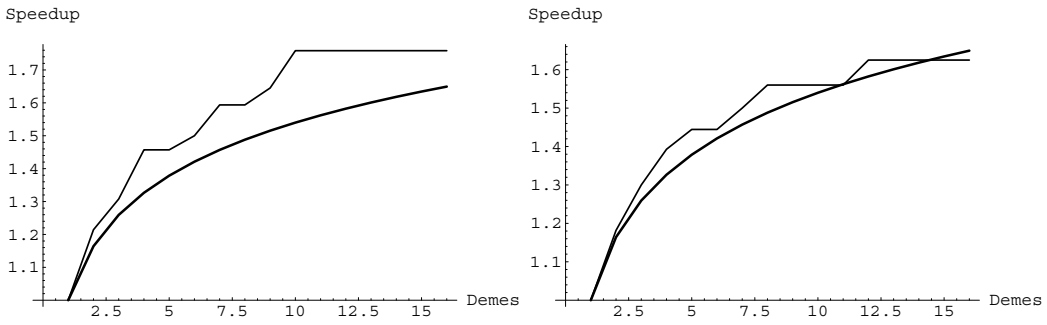
When the demes are isolated the communication time is null, and only the computation times are necessary to calculate the expected speedup. In every generation the GA evaluates all its population, and the number of generations until convergence may be assumed to be a domain-dependent constant, g. Holding the quality constant the speedup for isolated demes is

$$S_p = \frac{gn}{gn_d} = \frac{n}{n_d}. \tag{5.1}$$

Substituting the population sizes for the serial and isolated parallel GAs (equations 2.7 and 4.4, respectively) into equation 5.1 and simplifying terms gives a simple equation for the speedup (Cantú-Paz & Goldberg, 1997):

$$S_p = \frac{\ln(1 - \frac{\hat{Q}}{m})}{\ln(1 - \hat{P})}. \tag{5.2}$$

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



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

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

Figure 5.1: 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.

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 in this section are the average over 100 independent runs. The theoretical predictions for the parallel speedup along with the experimental results are plotted in figure 5.1. 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.

5.3 Fully Connected Demes

The speedup is expected to be greater in this case than with isolated demes, because the deme size required to reach a desired solution is substantially smaller. However, the cost of communications will partially offset the reduction in computation time.

The time that each deme uses to communicate with the other $r-1$ demes is $(r-1)T_c$, and the time used in computations is gn_dT_f, where g represents the number

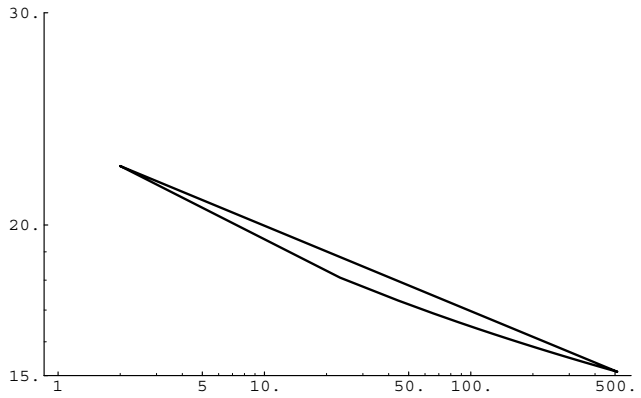


Figure 5.2: Charactering the parallel population size with a general power-law function. The thin line is the population size obtained with equation 4.12 and the bold line is the power-law model with adequately chosen constants.

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 may be estimated as

$$T_p = gn_dT_f + (r-1)T_c. \tag{5.3}$$

At this point we can observe that as more demes are used there is a tradeoff between increasing communication cost and decreasing computations (because smaller deme sizes are needed). This tradeoff entails the existence of an optimal number of demes that minimizes the parallel time. The rest of this section deals with finding the optimum.

To optimize the parallel time, we set $\frac{\partial T_p}{\partial r}$ to zero and solve to obtain the optimal number of demes. 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: $n = Ar^B$, 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 deme counts, and n_1 and n_2 are the corresponding parallel population sizes (obtained by using r_1 and r_2 in equation 4.12). The value for A can be obtained directly as $A = \frac{n_1}{r^B}$. Figure 5.2 shows an example of this approximation.

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 5.3 results in

$$T_p = gAr^BT_f + (r-1)T_c. \tag{5.4}$$

Now we can set the derivative of T_p to zero and solve for r to obtain the optimal number of demes as

$$r^* = \left(-\frac{ABgT_f}{T_c}\right)^{\frac{1}{1-B}} = (-ABg\gamma)^{\frac{1}{1-B}}, \tag{5.5}$$

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

The models of this section were validated with experiments using the same test functions of the previous section. Figure 5.3 shows the predictions and the experimental results of the parallel speedups using 4- and 8-bit trap functions. The parameters for the GAs were the same as for the experiments with isolated demes, and the same experimental procedure was used.

Both the predictions and the experimental results show an optimal speedup at

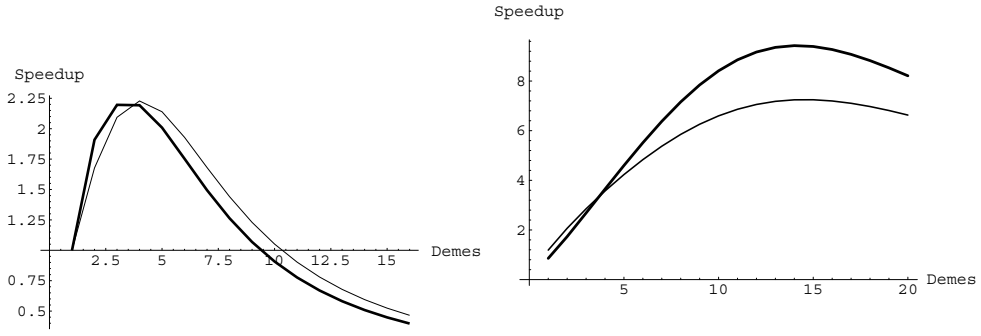
approximately the same number of demes. Note that for the 4-bit trap function, the speedup is less than one for several deme counts, which means that the parallel algorithm takes longer finish than the serial GA. This poor performance is mainly due to the expensive communications in our system, but also since the 4-bit problem can be solved with relatively small populations the savings on computation costs 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.

In contrast, the master-slave algorithm would perform poorly on these two problems because the computations are very simple. In the case of the 4-bit trap function, the single population size required to reach a solution with 80% of the BBs correct is 110 individuals. The optimal number of processors in a master-slave GA would be $\sqrt{n\frac{T_f}{T_c}} = \sqrt{110\ \frac{0.034}{19}} = 0.44$, which means that a configuration with a single processor would be the fastest. The situation would be slightly better in the case of the 8-bit trap, because the population required to reach the required solution is much larger than with the 4-bit trap (1450 individuals). For the 8-bit problem, the optimal number of processors would be $\sqrt{1450\ \frac{0.061}{19}} = 2.13$.

5.4 Summary

The first result of this chapter is that the parallel speedup is only modest when there is no communication between the demes. Although this may seem counterintuitive at first, the analysis shows that to reach solutions of the same quality, the size of the isolated demes is not very different from the population size of a simple GA.



(a) 4-bit trap function with 20 BBs. (b) 8-bit trap function with 10 BBs.

Figure 5.3: Predicted and experimental speedups for fully deceptive trap functions using 1 to 16 fully connected demes. The quality demanded was 80% BBs correct. The thick lines are the theoretical predictions and the thin lines are the experimental results.

On the other hand, when the demes communicate there are great improvements in performance. The analysis shows how to estimate the execution time of the parallel GA, and how to find the number of demes and a corresponding deme size that minimize it. Moreover, the formulas for the optimal deme size and deme count can be adjusted to consider particular implementations and to predict the effect of improvements in hardware or software in the performance of parallel GAs. With these tools, it is possible to guide the developer to those aspects of the implementation that will make a greater impact on performance.

Chapter 6

Markov Chain Models of Multiple Demes

This chapter extends the modeling of the search quality of parallel genetic algorithms with multiple demes. The chapter starts by considering the upper bounding case of topologies and migration rates that has been used in the previous two chapters. This case is the easiest to analyze, and it is used to introduce the methodology that will be applied to the more complex cases. The second algorithm considered in this chapter also has a fully connected topology, but the migration rate may take any value. Finally, the analysis is extended to arbitrary topologies that are more scalable and likely to be used in practice.

As in previous chapters, the migration frequency is bounded, and migration occurs after the demes converge. Then, the migrants are incorporated into the demes, and the algorithm restarts. The interval of time between migrations is called an *epoch*. Until now we have considered only the first two epochs but the analysis of this chapter considers any number of epochs.

The modeling is based on Markov chains, because the success of a demes depends solely on the events of the previous epoch. The objective is to predict the solution quality after each epoch and also in the long run. In addition, it is possible to calculate the expected number of epochs until all the populations converge to the same solution and no further improvement is possible.

The next two sections treat the fully-connected case. Section 6.1 examines the case where the migration rate is set to the maximum value possible, and then section 6.2 refines the analysis to consider lower migration rates. Experiments that

validate the accuracy of predictions are presented along with the theory. The models are extended in section 6.3 to consider arbitrary topologies and migration rates, and experiments that confirm the extended model are presented in section 6.4. Finally, section 6.5 summarizes the findings of this chapter and briefly discusses avenues of future research.

6.1 Fully Connected Demes with Maximum Migration Rates

Chapter 4 showed how to predict the quality of the solution at the end of the second epoch when the algorithm uses a maximal migration rate. The main idea was to estimate the number of copies of correct BBs in a deme after migration as $x_1 = n_d P_{bb}$, and to use this as the starting point of the gambler's ruin model.

This simple idea results in good predictions after the second epoch, and one would be tempted to generalize the model to multiple epochs by estimating the number of BBs at the start of the τ-th epoch ($\tau > 0$) as $x_\tau = n_d P_{bb}(x_{\tau-1})$. But unfortunately, the gambler's ruin model cannot be iterated after the second epoch because x_τ increases steadily, and after a few iterations the probability of converging correctly would be 1, regardless of the population size or the number of demes.

For example, consider a fitness function formed by concatenating 20 copies of a 4-bit fully deceptive trap function. Table 6.1 shows that after only eight (five) epochs, the iterated gambler's ruin model predicts that a parallel GA with a deme size of 16 (32) individuals will converge to the correct solution. In reality, the correct value of a partition may not appear in any of the demes, or it may be lost after some epochs, but none of this is taken into account in the simple iterated model.

6.1.1 Modeling with Markov Chains

To predict the quality of solutions found after an arbitrary number of epochs we need to determine accurately the starting point of the random walk. The number of correct

	Epoch	x_τ	$P_{bb}(x_\tau)$
$n_d = 16$	0	1	0.2144
	1	3.4	0.5668
	2	9	0.9024
	3	14.4	0.9894
	4	15.8	0.9990
	5	15.98	0.9999
	6	15.99	0.9999
	7	16	1
$n_d = 32$	0	2	0.3753
	1	12	0.9410
	2	30.11	0.9996
	3	31.99	0.9999
	4	32	1

Table 6.1: The iterated gambler's ruin model always converges to 1, regardless of the deme size (n_d) or the number of demes.

BBs at the start of an epoch depends directly—and solely—on how many demes converged correctly in the previous iteration. In particular, if i demes converged correctly and n_d is the deme size, each deme would start the current epoch with an average of $\chi_i = \frac{in_d}{r}$ copies of the correct BB, and the probability that it converges correctly is $P_{bb}(\chi_i)$. The problem consists on computing the number of demes i that have the right BB after each epoch.

The probability that a deme converges correctly the first time is $P_{bb}(x_0)$, where $x_0 = \frac{n_d}{2^k}$ is the expected number of BBs in a randomly initialized population. Since the demes evolved independently, at the end of the first epoch the distribution of demes with the right BB has a binomial distribution:

$$\mathbf{V}_1(i) = \binom{r}{i} P_{bb}^i(x_0) \left(1 - P_{bb}(x_0)\right)^{r-i}, \tag{6.1}$$

$\mathbf{V}_\tau(i)$ denotes the probability that exactly i demes converge correctly after the τ-th epoch. The probability of converging correctly after the second epoch is

$$P_{bb_2} = \sum_{i=0}^{r} \mathbf{V}_1(i) \cdot P_{bb}(\chi_i), \tag{6.2}$$

which can be generalized and expressed as the vector product

$$P_{bb_\tau} = \mathbf{V}_{\tau-1}\mathbf{U}, \tag{6.3}$$

where $\mathbf{U}(i) = P_{bb}(\chi_i)$. The challenge is to calculate the distribution of correct demes after an arbitrary number of epochs ($\mathbf{V}_\tau$), and for this purpose we use Markov chains. The states of the chain represent the number of demes that converged to the correct BB in a given epoch. The transition matrix is defined with the probabilities of going from a state with i demes correct to a state with j demes correct as follows:

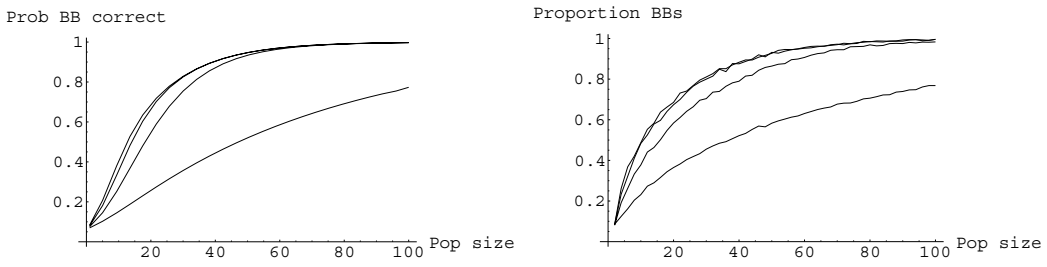
$$\mathbf{M}(i,j) = \binom{r}{j} (P_{bb}(\chi_i))^j (1 - P_{bb}(\chi_i))^{r-j}. \tag{6.4}$$

The distribution of the number of demes that converge correctly after τ epochs is given by

$$\mathbf{V}_\tau = \mathbf{V}_1 \mathbf{M}^{\tau-1}, \tag{6.5}$$

and the probability of converging to the correct BB may be calculated with equation 6.3.

Figure 6.1 presents an example of the predictions of equation 6.3 after several epochs on four fully connected demes. The function used in the example is the same 20-BB 4-bit trap problem used in previous chapters. The GA uses pairwise tourna-



(a) Theoretical predictions

(b) Experimental results

Figure 6.1: Probability of converging to the correct BB after 1,2,3,and 4 epochs (from right to left).

ment selection, two-point crossover with probability 1, and no mutation. Contrast the predictions with the iterated gambler's ruin example in table 6.1. Note that the major improvement in quality comes at the second epoch, and therefore the calculations derived in the previous chapters are very significant for the design of parallel GAs.

6.1.2 Parallel Demes in the Long Run

The example above also suggests that the probability of finding the correct BB converges to a fixed value after a few epochs. Another application of Markov chains is to calculate the long run distribution of the number of demes that find the BB, that is $\lim_{\tau\to\infty} \mathbf{V}_\tau$. Substituting this distribution in equation 6.3 gives the probability that in the long run the parallel GA finds the correct BB. The remainder of this section treats this issue, and also shows how to calculate the expected number of epochs until all the demes converge to the same solution.

First, recall the assumption made in chapter 2 that crossover and mutation do not create or destroy significant numbers of correct BBs. This assumption implies that if the correct BB disappears from all the demes there is no way of recovering it. Likewise, when all the demes converge to the correct BB there is no chance of losing it. These facts are reflected in the transition matrix: the first row (corresponding to

state 0, when no deme has the correct BB) is 1,0,0,...0, and the last row (state r, when all the demes have the correct BB) is 0,0,...,1. The states 0 and r are called absorbing or persistent states, and since there are no possible transitions between them, the chain has two closed absorbing sets. All the other states in the chain are called transient states.

The fundamental matrix method (Isaacson & Madsen, 1976) may be used to calculate the distribution of demes with the correct BB in the long run and the expected number of epochs until absorption. To use this method the states need to be reordered, and the transition matrix rewritten as

$$\mathbf{M} = \begin{pmatrix} \mathbf{P}_1 & 0 & 0 \\ 0 & \mathbf{P}_2 & 0 \\ \mathbf{R}_1 & \mathbf{R}_2 & \mathbf{Q} \end{pmatrix},$$

where $\mathbf{P}_1$ and $\mathbf{P}_2$ are the submatrices with the transition probabilities within the two closed persistent sets, which in our case consist of a single state each (therefore, $\mathbf{P}_1 = \mathbf{P}_2 = 1$); $\mathbf{Q}$ is a submatrix with the transition probabilities within the transient states; and $\mathbf{R}_1$ and $\mathbf{R}_2$ contain the probabilities of going from each transient state to each of the persistent states.

The expected absorption time from each transient state i is given by the i-th element of

$$\mathbf{T} = \mathbf{N1},$$

where the matrix $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$ is called the fundamental matrix, $\mathbf{I}$ is the identity matrix, and $\mathbf{1}$ is a column vector of ones. Augment $\mathbf{T}$ with a zero at the beginning and a zero at the end to account for the expected absorption times from state 0 and state r, respectively. Now the mean time until absorption may be calculated by multiplying the initial distribution of demes with the correct BB (given by equation 6.1) by the extended $\mathbf{T}$ ($\mathbf{T}'$) as follows:

$$\langle \tau \rangle = \mathbf{V}_1 \mathbf{T}'. \tag{6.6}$$

The absorption probabilities from the transient state i to the persistent state l is given by the (i, l)-th entry of $\mathbf{NR}$, where $\mathbf{R}$ is the matrix formed with the elements of $\mathbf{R_1}$ and $\mathbf{R_2}$. Recall that $\mathbf{P_2}$ is the submatrix that contains state r, which represents the case where all the demes converge to the right BB. Therefore, the distribution of probabilities, $\mathbf{A}$, of being absorbed into state r is given by the second column of $\mathbf{NR}$. Augment $\mathbf{A}$ with a zero at the beginning and a one at the end, to account for the chances of being absorbed from state 0 and state r, respectively. To find the mean probability of being absorbed at state r, multiply the initial distribution of demes with the correct BB by the extended vector $(\mathbf{A}')$:

$$P_{bb_\infty} = \mathbf{V}_1 \mathbf{A}'. \tag{6.7}$$

Figure 6.2 presents plots of P_{bb_∞} using four demes with $0 \leq n_d \leq 100$ individuals each, and a plot of P_{bb} (equation 2.5) using a population size of $n = 4n_d$. The two plots overlap perfectly, suggesting that the probability that r fully connected demes of size n_d converge correctly in the long run is the same as the probability of success of a GA with a single population with rn_d individuals. This is important because it suggests that *in the long run* the solution's quality does not degrade or improve when a population is partitioned into smaller fully connected demes that communicate with the maximum migration rate possible.

It is not immediately clear what would be the outcome if the populations communicate using a lower migration rate or a different topology. The following sections study the effects of these two parameters on the quality of the solutions.

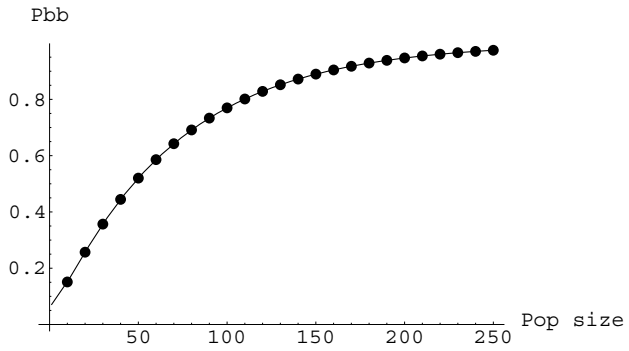


Figure 6.2: In the limit, a parallel GA with r fully connected populations using a maximal migration rate (dots) has the same chance of finding the solution as a simple GA with an aggregate population (continuous line).

6.2 Arbitrary Migration Rates

Using the maximal migration rate simplifies the calculation of the number of BB present in a deme after each epoch, because the contribution from all the demes is uniform. This section extends the calculations to cases with lower migration rates. The migration rate is denoted by ρ and represents the fraction of the population that migrates. The method used here is very similar to the one used in the previous section, but with lower migration rates the initial number of BBs in a particular deme depends greatly on whether the local deme converged correctly in the previous epoch.

For example, consider that in a given epoch two out of three demes of a parallel GA converge correctly, and suppose that each population sends $\rho = 0.05$ of its individuals to the other two. At the start of the next epoch there are two possibilities for a particular deme. First, if it converged correctly in the previous epoch, then 95% of its individuals have the correct BB (90% were already there, and it obtained 5% from the other correct deme). On the other hand, if the deme did not converge correctly, only 10% of the population would have the correct BB, contributed by the other two demes.

To reflect this situation, the Markov chain needs twice as many states as before.

For each number of demes that converged correctly, the chain needs two states: one to represent the case when the major fraction of the population contains the BB (because the population converged correctly in the previous iteration), and another state to represent when the major fraction of the population is incorrect. A convenient way to order the states is that states 0 to $r-1$ represent the cases where the major fraction of the deme is incorrect, and states r to $2r-1$ represent the cases where major fraction is correct. This ordering of the states is arbitrary, and any other ordering would be adequate. As before, there is one absorbing state for the case when all the demes converge incorrectly (state 0), and there is one state for when all the demes converge to the correct BB (state $2r-1$). The rest are transient states.

The initial distribution is

$$\mathbf{V}_1(i) = \begin{cases} \binom{r-1}{i}[P_{bb}(x_0)]^i\ [1-P_{bb}(x_0)]^{r-i} & \text{if } i < r, \\ \binom{r-1}{i-r}[P_{bb}(x_0)]^{i-r+1}\ [1-P_{bb}(x_0)]^{2r-i-1} & \text{if } i \geq r, \end{cases} \tag{6.8}$$

for $i \in [0, 2r-1]$. The transition matrix becomes

$$\mathbf{M}(i,j) = \begin{cases} \binom{r-1}{j}[P_{bb}(\chi_i)]^j\ [1-P_{bb}(\chi_i)]^{r-j} & \text{if } j < r, \\ \binom{r-1}{j-r}[P_{bb}(\chi_i)]^{j-r+1}\ [1-P_{bb}(\chi_i)]^{2r-j-1} & \text{if } j \geq r, \end{cases} \tag{6.9}$$

where χ_i is the starting point of the random walk for each state i, and it depends on the migration rate ρ and on how many demes converged correctly:

$$\chi_i = \begin{cases} n_d\, i\, \rho & \text{if } i < r, \\ n_d\, ((i-r)\rho + 1 - (r-1)\rho) & \text{if } i \geq r. \end{cases} \tag{6.10}$$

The probability of converging to the correct BB after τ epochs is given by $P_{bb_\tau} = \mathbf{V}_{\tau-1}\mathbf{U}$ (equation 6.3), where $\mathbf{U}(i) = P_{bb}(\chi_i)$, and χ_i is defined as above.

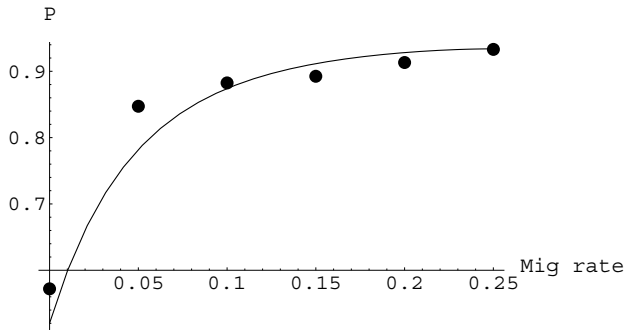


Figure 6.3: The probability of converging to the BB increases with higher migration rates. The theoretical predictions (continuous line) are compared against experimental results.

Figure 6.3 illustrates the probability of reaching the correct solution as a function of the migration rate. The probability of success increases rapidly with higher migration rates. The example uses four fully-connected demes with 50 individuals each; the test function is a 20-BB 4-bit trap problem; and only the first two epochs are considered.

As before, the fundamental matrix method may be used to predict the long-term behavior of the parallel GA with arbitrary migration rates. The states have to be reordered as in the previous section, and the calculations are similar. Figure 6.4 shows the probability that in the long run the parallel GA will converge to the correct solution as a function of the migration rate. The plot suggests that only a moderate migration rate is sufficient to reach the same solution as a simple GA with a aggregate population.

Since all the individuals are the same when migration occurs, there are no cost penalties associated with higher rates: only one individual needs to be sent and it can be replicated any number of times at the receiving deme.

6.3 Arbitrary Topologies

The previous two sections showed how the complexity of the analysis increased when the algorithm became more flexible. In the first case, when the migration rate was

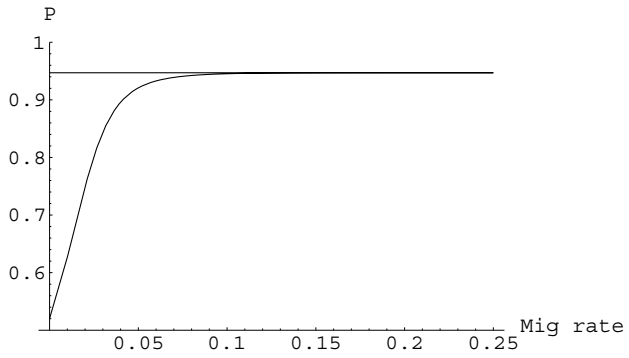


Figure 6.4: The long-run probability of converging to the correct BB increases rapidly with higher migration rates. The example considers four fully connected demes, with 50 individuals each, working on a 20-BB 4-bit trap function. The horizontal line is the probability that four fully connected demes with maximal migration (or a simple GA with 200 individuals) will eventually converge to the right BB.

maximal, the only information required to calculate the number of BBs at the start of an epoch was the number of demes that converged correctly. When the migration rate was allowed to change, additional states were required to represent whether the local deme had converged correctly or not. The nature of the information represented by each state changed from a mere count of demes correct to include limited spatial information (i.e., it became important to know *which* demes had the correct BB).

More spatial information is required when considering arbitrary topologies. In this case, it is not sufficient to know how many demes converged correctly in the previous epoch, but also *exactly* which ones. This information is represented in the states of the Markov chain, and therefore we expect to use many additional states. Since a deme can either have the BB or not, in a setup with r demes there are 2^r possible states. A natural representation of these information is to use a binary string s_i of length r for each state i. The k-th bit of the string, $s_i(k)$, corresponds to the k-th deme and is set to one if the deme converged correctly and to zero otherwise. The states of the Markov chain are numbered from 0 to $2^r - 1$ and can be conveniently labeled with the integers represented by the strings. For example, in a configuration with 8 demes, state 10 corresponds to the string $s_{10} = 00001010$ and represents the

case where demes 1 and 3 converged correctly.

The construction of the transition matrix is not as straightforward as when the demes are fully connected. As before, the core of the modeling is to determine how many copies of the BB are present in a deme just after migration. The first step is to determine how many of the neighbors have the right BB, and for this we use a topology-dependent neighborhood function $N(a)$ that returns a set $\mathcal{N}_a$ with the indices of the neighbors of deme a. For example, in a bidirectional ring the neighbors of deme 1 are demes 0 and 2 (i.e., $\mathcal{N}_1 = N(1) = \{0, 2\}$). Since the states contain the information about which demes have the correct BB, it is easy to determine how many neighbors of a deme a converged correctly when the chain is at state i as

$$c_{i,a} = \sum_{k \in \mathcal{N}_a} s_i(k). \tag{6.11}$$

With this information, the number of BBs in deme a at the start of the epoch may be calculated as

$$\chi_{i,a} = n_d \left[c_{i,a}\, \rho + s_i(a) \left(1 - \delta\rho\right) \right], \tag{6.12}$$

where δ is the degree of the topology and ρ is the migration rate. The first term above, $c_{i,a}\, \rho$, is the fraction of the deme with the correct BB that is contributed by the neighbors. The second term is about the individuals that were already present in the deme. When $s_i(a)$ equals one, deme a converged correctly in the previous epoch, and the fraction $1 - \delta\rho$ of the population that remained unchanged after migration contains the correct BB. If $s_i(a) = 0$, then the only correct BBs in the deme come from its neighbors.

The probability that deme a will converge correctly is given by $P_{bb}(\chi_{i,a})$, and so

the probability of going from state i to state j is given by

$$\mathbf{M}(i,j) = \prod_{a=1}^{r} \left[s_j(a) P_{bb}(\chi_{i,a}) \; + \; (1 - s_j(a))(1 - P_{bb}(\chi_{i,a})) \right]. \tag{6.13}$$

For each value of a, only one term of the equation above is different than 0, depending on whether the a-th deme is correct or incorrect in state j.

The distribution of states is given by $\mathbf{V}_\tau = \mathbf{V}_1 \mathbf{M}^{\tau-1}$ (equation 6.5), and the probability of converging correctly is determined by equation 6.3 ($P_{bb_\tau} = \mathbf{V}_{\tau-1}\mathbf{U}$). However, now

$$\mathbf{U}(i) = P_{bb}(\chi_{i,a}), \tag{6.14}$$

and $\chi_{i,a}$ is defined in equation 6.12. Any choice of a may be used in this equation when the demes are connected by a regular topology, because on average all demes will have the same outcome.

The initial distribution of states is given by

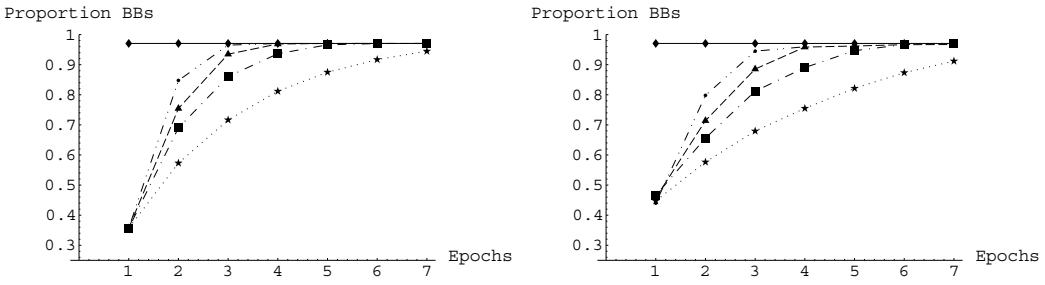
$$\mathbf{V}_1(i) = P_{bb}(x_0)^{u(s_i)} (1 - P_{bb}(x_0))^{r - u(s_i)}, \tag{6.15}$$

where $u(s)$ is a function that counts the bits set to one in the string s.

The long run probability of success may be found by reordering the states and using the fundamental matrix method, as was described in section 6.1.

6.4 Experiments

Several experiments were conducted to assess the accuracy of the model described in the previous section. The experiments used eight demes and the test function was the



(a) Theory.

(b) Experiments.

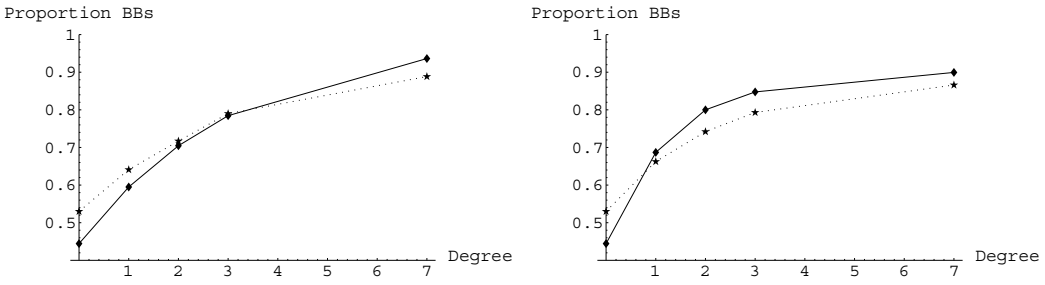
Figure 6.5: Solution quality after several epochs. The graphs include data for uni- and bi-directional rings, a hypercube, and a fully connected topology (from bottom to top, respectively). The horizontal line in both graphs is the prediction of the quality that would be reached by a simple GA with an aggregate population.

same 20-BB 4-bit trap used previously. The GAs used pairwise tournament selection, 2-point crossover with probability one, and no mutation. The results presented are the average over 100 repetitions.

The first set of experiments was designed to compare the quality of the solutions reached using different topologies. Figure 6.5 shows plots of the quality versus the number of epochs for a fully connected topology and three topologies frequently used by practitioners: a uni-directional ring, a bi-directional ring, and a hypercube. Each of the eight demes has 30 individuals, and the migration rate was set to the maximum value possible for each topology: 50% for the uni-directional ring, 33% for the bi-directional ring, and 25% for the hypercube.

In all cases, the quality increases as more epochs are used, but the rate of increase depends on each topology. The uni-directional ring needs the most epochs to reach the highest quality possible—which is the same quality that simple GA with an aggregate population would reach—while the fully connected topology realizes its full potential using the fewest epochs.

The experiments above suggest that topologies where the demes have more neigh-



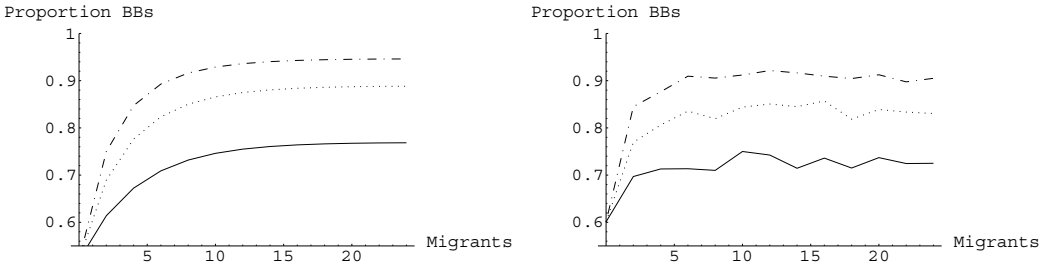
(a) 10% migration per deme.

(b) 20 migrants total.

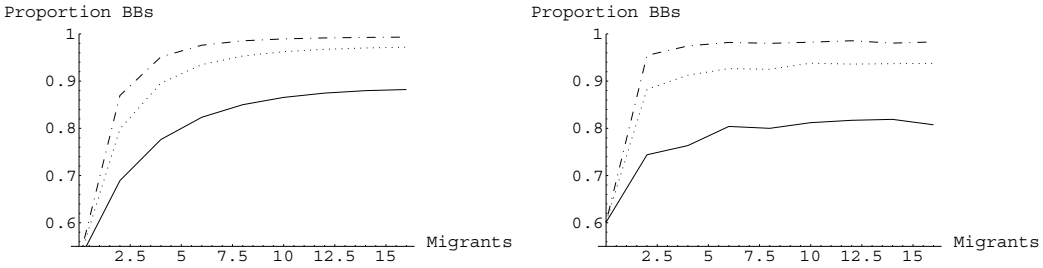
Figure 6.6: Solution quality as a function of the degree of the topology. The continuous lines are the theoretical predictions, and the dashed lines are the experimental results.

bors reach solutions of higher quality than sparse topologies: at any given epoch the fully connected topology reached the best solutions, while the uni-directional ring reached the worst. To visualize the effect of the degree more clearly, figure 6.6 shows experiments where the degree varies while the deme size and migration rate are constant. The experiments used the four topologies used before (with degrees 1, 2, 3, and 7). The experiments considered eight demes with 40 individuals each, and the quality was measured at the end of the second epoch. In the first experiment the migration rate was set to 10% per deme (so the number of migrants received by a deme increases with the degree), and in the second experiment the total number of migrants was set to 20.

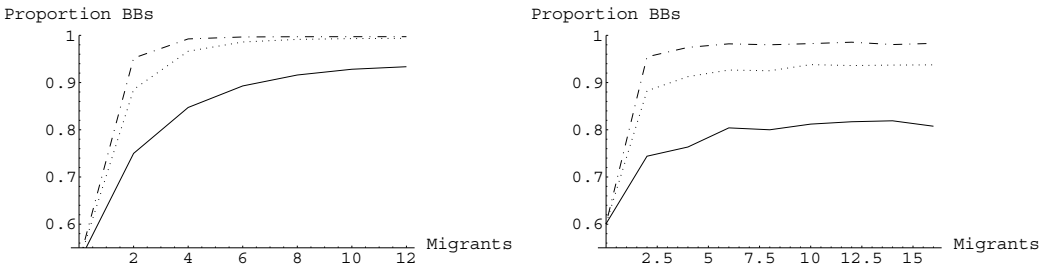
In the two previous experiments, the quality increased quickly as the topologies became denser, but the marginal improvements are smaller. This is important because topologies with higher degrees have higher communication costs, and the marginal increase in quality may not be large enough to justify the added cost. The next chapter explores this issue in detail, and shows how to calculate the optimal degree that minimizes the execution time, while reaching a solution of a predetermined quality.



(a) Uni-directional ring.



(b) Bi-directional ring.



(c) Hypercube.

Figure 6.7: Quality versus number of migrants. Each graph shows the quality using multiple migration rates after two, three, and four epochs (from bottom to top in each graph). Graphs on the left are theoretical predictions, and graphs on the right are experiments.

The next set of experiments was designed to test the accuracy of the models using varying migration rates. The experiments used rings (uni- and bi-directional) and a hypercube to connect eight demes with 50 individuals each. The migration rate was varied from zero to the maximum rate possible in each topology, and the quality of the solutions was recorded at the end of the second, third, and fourth epochs. (During the first epoch the demes are isolated and their performance is equivalent to a migration rate of zero.) The results shown in figure 6.7 confirm that the model accurately predicts the quality over a variety of migration rates and topologies.

Based on the results of the previous sections, higher migration rates are expected to produce better solutions. The experiments of this section confirm that this is indeed the case, and that the quality increases rapidly with higher migration rates. Also, the uni-directional ring reached the lowest-quality solutions, while the hypercube found the best.

6.5 Summary

This chapter presented models that predict the expected solution quality of parallel GAs with multiple populations after any number of epochs and for any choice of number of demes, deme size, topology, or migration rate. The modeling used Markov chains to determine the number of correct BBs present in the demes at the beginning of each epoch. Then, the gambler's ruin model was used to predict the quality of the solutions. The size of the Markov chains increased progressively as more flexibility was allowed in the algorithms. The first algorithm examined was an upper bound on topologies and migration rates. This case was the simplest to examine because the contributions from all the demes are the same, and only r states are required. The second case still considers a fully connected topology, but it was extended to arbitrary migration rates. This case required additional states to represent the varying contributions from the local deme and its neighbors. Finally, the model was extended to

arbitrary topologies and migration rates, and the number of states increased again. The accuracy of the models was tested with computational experiments using one fully deceptive test function and varying all the parameters of interest. In all cases, the predictions match closely the observed quality of the solutions after any number of epochs.

This chapter extended our understanding of multi-deme parallel GAs in several important directions. First, we observed that the greatest gain in quality always occurs after the second epoch. This is significant because it reinforces the significance of the calculations of the two previous chapters that showed how to derive closed-form expressions for the deme sizes and how to manipulate these expressions to determine the configurations that minimize the execution time.

Another contribution is that the models and experiments presented in this chapter suggest that in the long run the parallel GAs reach solutions of the same quality as a simple GA with a population equivalent to the aggregate of the demes. Partitioning the population does not degrade or improve the solution quality as long as the migration rate is not very low. Furthermore, it appears that only a few epochs are necessary to reach the same solution as the simple GA.

Another important observation is that the quality improves with higher migration rates, regardless of the topology. In the algorithm examined here there is no reason to use low migration rates. Since migration occurs after the demes converge, the cost of communications is independent of the migration rate: only one individual has to be sent and it can be replicated as necessary in the receiving deme.

The calculations and experiments also suggest that higher network degrees improve the quality of the search. However, higher degrees raise the cost of communications, and result in a tradeoff between increasing the quality or making the algorithm slower. In many situations there is a minimal desired solution quality, and when the parallel GA is required to reach it, there is an optimal configuration that minimizes the total execution time. The next chapter addresses these issues in detail.

Chapter 7

Topologies, Migration Rates, and Scalable Parallel GAs

Previous chapters examined bounding cases of multiple-deme parallel GAs. The investigation of the extreme cases resulted in useful design guidelines for parallel GAs, but those bounding configurations are not commonly used by practitioners because neither of them is scalable. On one hand, when the demes are isolated the improvements in quality are marginal and the total computational cost grows very fast. On the other extreme where the demes are fully connected, the cost of communications quickly becomes impractical as more demes are used.

This chapter addresses the problem of scalable communications. Its goal is to find the configuration of multiple demes that reaches the desired solution in the shortest time possible. As in the study of bounding cases, the analysis considers the issues of cost and solution quality simultaneously.

The models assume that all the demes have the same number of neighbors, but there are no other restrictions on the topology or on the migration rates. As with the bounding cases, the research focuses initially on the first two epochs, and it is later extended to any number of epochs.

The chapter is organized into four sections as follows. Section 7.1 describes how the deme size, the migration rate, and the degree of the connectivity graph affect the chances that the desired solution is reached after the second epoch. As in previous chapters, the analysis shows how to find a configuration that minimizes the execution

time. This section also revisits the fully connected topology, and shows that its optimal configuration is a good approximation of the optimal times of sparser topologies. Section 7.2 generalizes the results to multiple epochs. It shows that the solutions reached by different topologies with the same degree are almost identical, and then it focuses on one family of topologies to determine how the quality improves after arbitrary epochs. Then, the results of the modeling are used to find a configuration that minimizes the execution time. Section 7.3 estimates the number of epochs until all the demes converge to the same solution, and determines the optimal number of demes. Finally, section 7.4 summarizes the findings of this chapter.

7.1 Degree of Connectivity

An important property of the connectivity graph between the demes is its *degree*, which is the number of neighbors of each deme. This chapter 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 analyzes 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 of this section consider only the first two epochs of the algorithm, because closed-form expressions may be derived easily. The next section extends the modeling to multiple epochs.

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 4.3). 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 to reach $\hat{P}$ at the end of the second epoch we may use the solution of the gambler's ruin problem (eq. 2.6). Making

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

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

$$\widehat{x_1} = \frac{\ln(1-\hat{P})}{\ln\left(\frac{q}{p}\right)}. \tag{7.1}$$

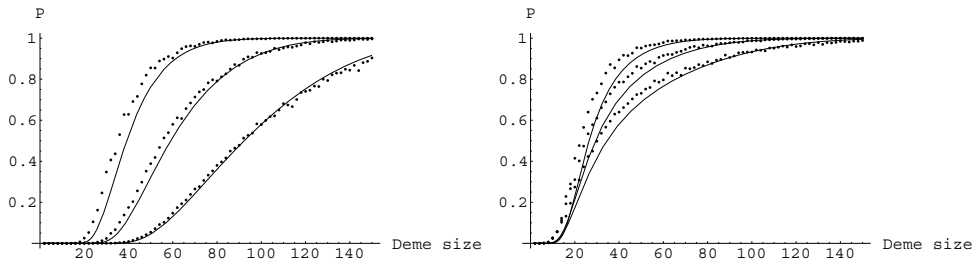
The next step is to determine the probability that a deme receives at least $\widehat{x_1}$ BBs from its δ neighbors. The probability that one neighbor sends the right BB is the same probability that it converged correctly in the first epoch, and is given by P_{bb} (eq. 2.6). Assuming that all the neighbors 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:

$$P_{x_1} = 1 - \sum_{i=0}^{\hat{\delta}-1} \binom{\delta}{i} P_{bb}^i (1-P_{bb})^{\delta-i}, \tag{7.2}$$

which can be approximated as

$$P_{x_1} = 1 - \Phi\left(\frac{\hat{\delta} - \delta P_{bb}}{\sqrt{\delta P_{bb}(1-P_{bb})}}\right), \tag{7.3}$$

where Φ denotes the CDF of a normal distribution with zero mean and standard deviation of one. Figure 7.1 shows the results of experiments that illustrate the accuracy of P_{x_1}. The test problem used in this experiments is a 20-BB 4-bit trap function, and the figure shows the average of 100 independent runs.



(a) 10% migration (b) maximal migration

Figure 7.1: Plot of the probability 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 1, 2, and 4 neighbors (from right to left in each graph) using different migration rates.

With higher migration rates, fewer neighbors must contribute the right 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 the results of the previous chapter that showed that the solution's quality increased with higher migration rates.

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 (see figure 7.2). Configurations with large demes and few neighbors have the same chance of success than some configurations with smaller demes but with more neighbors. This is the usual tradeoff between computation and computations: smaller demes require more neighbors to succeed. We would like to

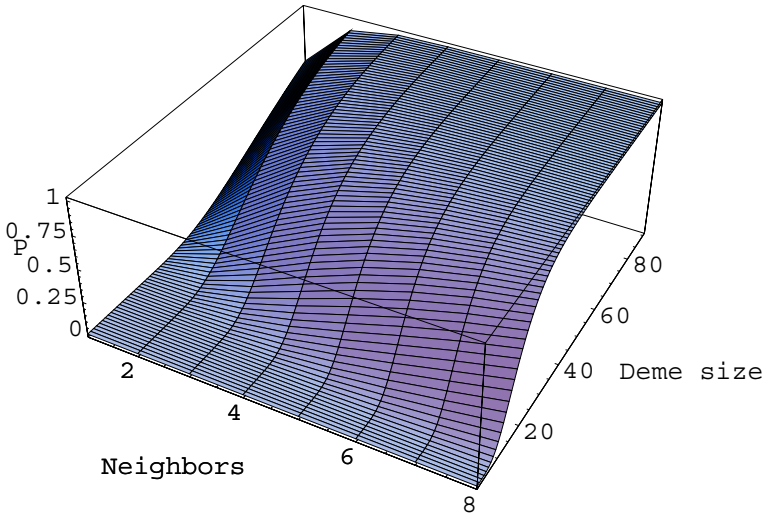


Figure 7.2: Plot of the probability of success with different configurations of deme sizes and number of neighbors. The migration rate in this example is 10%.

use the configuration that achieves the desired objective with the minimum cost.

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

$$T_p = g n_d T_f + \delta T_c, \tag{7.4}$$

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.

7.1.1 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

$$P_{x_1} = 1 - (1 + \exp(-1.6z))^{-1}, \tag{7.5}$$

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:

$$n_d = \frac{2^{k-2}}{\sqrt{\delta}} \frac{\hat{z} + \sqrt{\hat{z}^2 + \frac{c\widehat{x_1}}{\rho 2^{k-2}}}}{c}. \tag{7.6}$$

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:

$$n_d = \frac{n_0}{\sqrt{\delta}}. \tag{7.7}$$

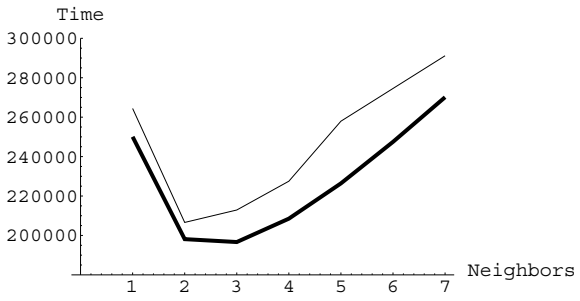


Figure 7.3: Comparison of theoretical (thick line) and experimental (thin line) execution times (in microseconds) of eight demes connected by topologies with different degrees.

Now, the total execution time as given by equation 7.4 may be easily optimized with respect to δ by making $\frac{\partial T_p}{\partial \delta} = 0$ and solving for δ:

$$\delta^* = \left(\frac{g n_0 T_f}{2 T_c} \right)^{2/3}, \tag{7.8}$$

and the optimal deme size can be found by substituting δ^* in equation 7.6.

Figure 7.3 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 $g = 50$. The figure shows the average of 100 runs using pairwise tournament selection, two-point crossover with probability 1.0, and no mutation. The migration rate is $\rho = 0.1$.

7.1.2 Fully Connected Topologies Revisited

Sometimes the magnitude of optimal degree will be greater than the number of demes, because it depends on the ratio of computation to communications γ, 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 alternate expression for the optimal number of demes in a fully connected topology. The idea is simply to realize that 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 demes,

$$r^* = \delta^* + 1. \tag{7.9}$$

Figure 7.4 has an example with the 20-BB 4-bit trap function used before. The example considers the same computing environment used in the experiments with fully connected demes in chapter 5. The fitness evaluation time is $T_f = 0.034$ ms, the communication time is $T_c = 19$ ms, and $g = 50$ generations are required to converge. Only the first two epochs are considered to facilitate comparisons with the results of chapter 5. 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. The optimal number of demes in the example ($r^* = 3.5$ that should be rounded to 4) agrees with the result in chapter 5, where the optimum was found to be four demes.

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

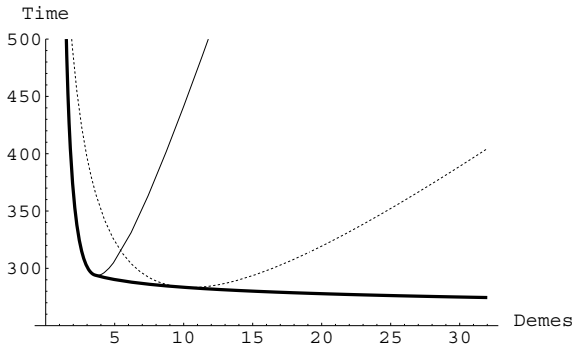


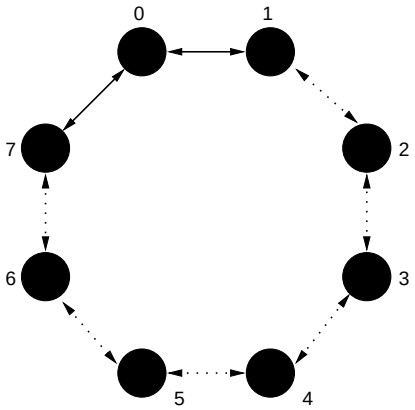
Figure 7.4: The execution time using the optimal degree decreases very slowly (bold line). It is bounded (and well approximated) by the optimal time of the fully connected topology (thin line). (See the text for an explanation of the points at $r = 2, 3$). The execution time of a topology of degree $r/4$ (dotted line) is plotted as an example.

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.

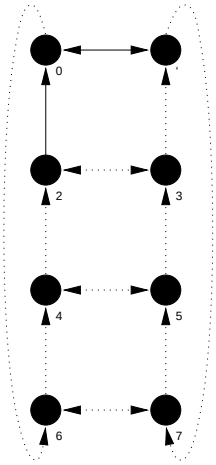
7.2 Considering Multiple Epochs

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. But if more than two epochs are used, a deme would receive 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.

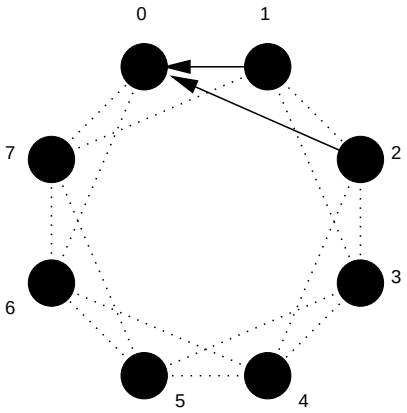
As an example of the issues involved, consider the topologies with degree $\delta = 2$



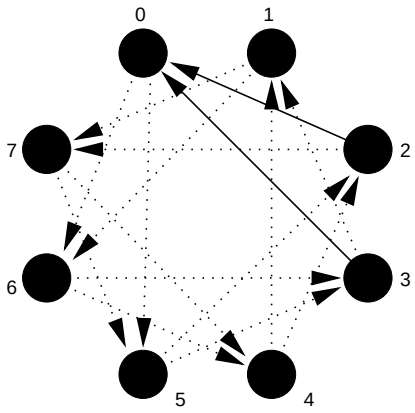
(a) Bi-directional ring.



(b) Ladder.



(c) +1+2 topology.



(d) +2+3 topology.

Figure 7.5: Different topologies with two neighbors.

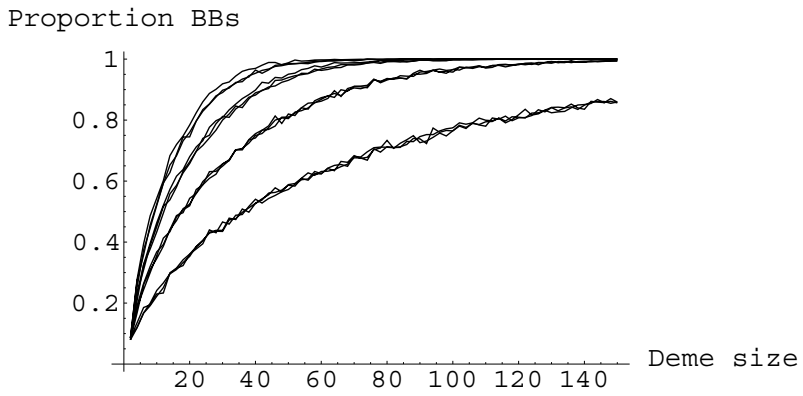
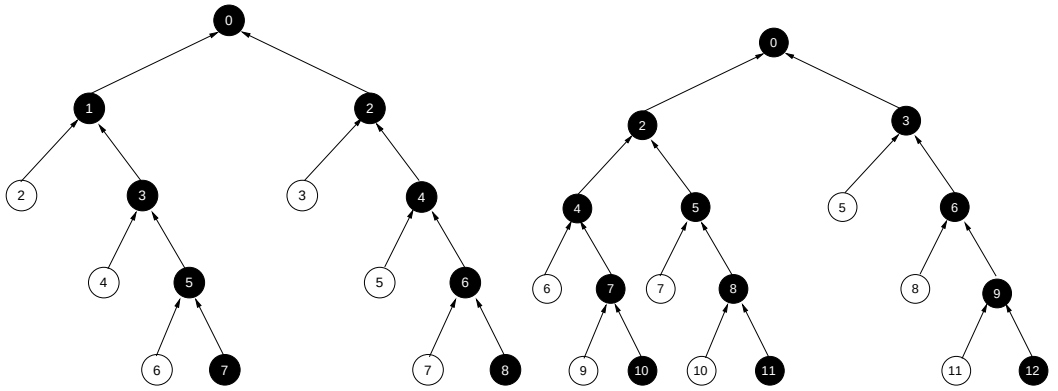


Figure 7.6: Average quality per deme after one, two, three, and four epochs (from right to left) using eight demes connected with different topologies of degree two.

depicted in figure 7.5. These are only four of the $\binom{7}{2} = 21$ possible topologies of degree two. Figure 7.6 shows the results of experiments with a 20-BB 4-bit trap function on eight demes connected with the four topologies. 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. The figure 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, as in the previous chapter, we can observe that the largest increase occurs after the second epoch. As one would expect, the results for different topologies during 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 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.



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

Figure 7.7: Tree representations of the extended neighborhoods of demes in two different topologies of degree 2 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 neighborhood, and they are not expanded to avoid clutter in the graphs.

7.2.1 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 7.2.1 shows two such trees corresponding 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

[1]For the mathematically inclined, 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. $r_\tau = |R_\tau|$.

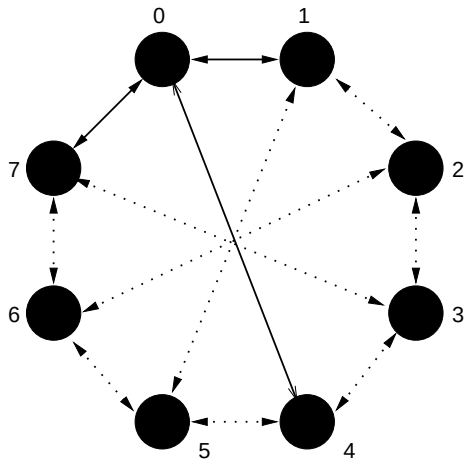


Figure 7.8: The cartwheel topology is similar to a bi-directional ring, but there is an extra link to the opposite deme. The diameter is $D = r/4$, and the degree is $\delta = 3$.

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 $c_m < 1$, and the quality becomes $P_{bb}(c_m r_\tau n_d)$.

In any case, the simple approximation explains why those topologies where more demes contribute at each epoch reach slightly better results. To verify the hypothesis, experiments were performed with two topologies where the same number of demes contribute after each epoch (r_τ is the same for both topologies for all values of τ). Eight demes were connected as an hypercube and a "cartwheel" (see figure 7.8). After four epochs, the two topologies yielded identical results, which are plotted in figure 7.9.

7.2.2 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

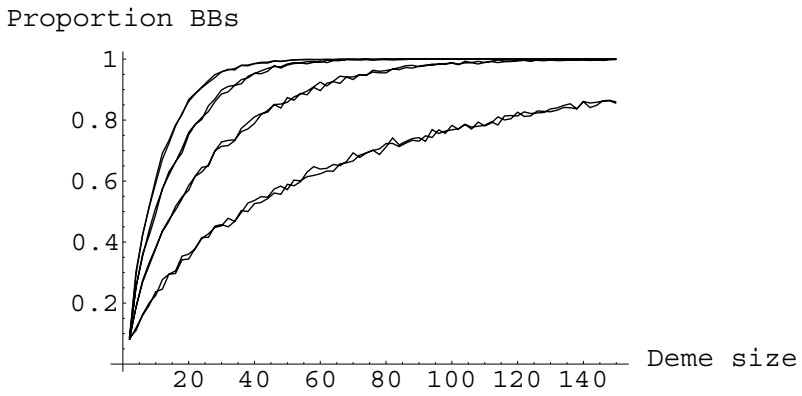


Figure 7.9: Average quality per deme after one, two, three, and four epochs using eight demes connected by two topologies of degree three. The quality after all epochs is identical because the size of the extended neighborhoods are the same in both topologies.

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 7.5 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 derived previously, the quality after τ epochs is $P_{bb_\tau} = P_{bb}(r_\tau n_d c_m)$. For simplicity, we use $n_\tau = r_\tau n_d c_m$ 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 7.7). Putting everything together, we may write n_τ as:

$$n_\tau = (\sqrt{\delta}\tau' + 1)n_d, \tag{7.10}$$

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 experimental setup as previous experiments in this chapter. The deme sizes were varied, and the quality was measured at the end of the first four epochs. Figure 7.10 shows that the predictions of $P_{bb}(n_\tau)$ match very well the experimental results.

We may also compare the predictions of $P_{bb}(n_\tau)$ with the Markov chains model derived in the previous chapter. Figure 7.11 presents the same quality predictions as in figure 7.10, and shows that for the +1+2 topology the difference between the two models is small. Another interesting test is to compare the two models on different topologies. Figure 7.12 compares the expected quality reached by eight demes of size 30 connected in three configurations: a uni-directional ring, a bi-directional ring, and a fully connected topology. The migration rate is set to the maximal value possible in each topology, and the first seven epochs are considered. Again, the differences between the two models are small, and it appears that $P_{bb}(n_\tau)$ may be used confidently as a good predictor of solution quality.

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

$$n_d = \frac{1}{\sqrt{\delta}\tau' + 1} \frac{2^k \ln(1-\hat{P})}{\ln \frac{q}{p}}, \tag{7.11}$$

which can be rewritten in a much compact form by grouping all the problem-dependent

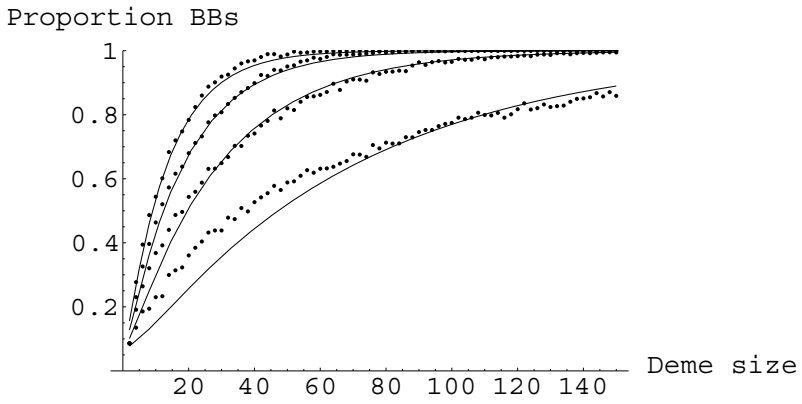


Figure 7.10: Theoretical predictions (line) and experimental results (dots) of the average quality per deme after 1, 2, 3, and 4 epochs (from right to left) using eight demes connected by a +1+2 topology.

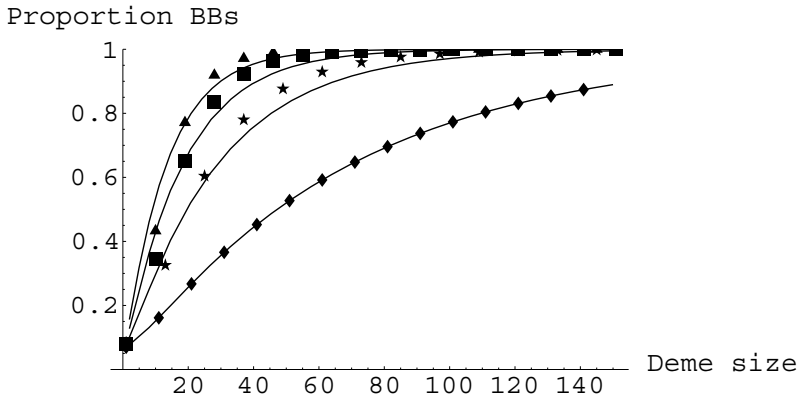


Figure 7.11: Comparison of the approximate (continuous lines) and the Markov chains models (dots). The graph shows the average quality per deme after 1, 2, 3, and 4 epochs (from right to left) using eight demes connected by a +1+2 topology.

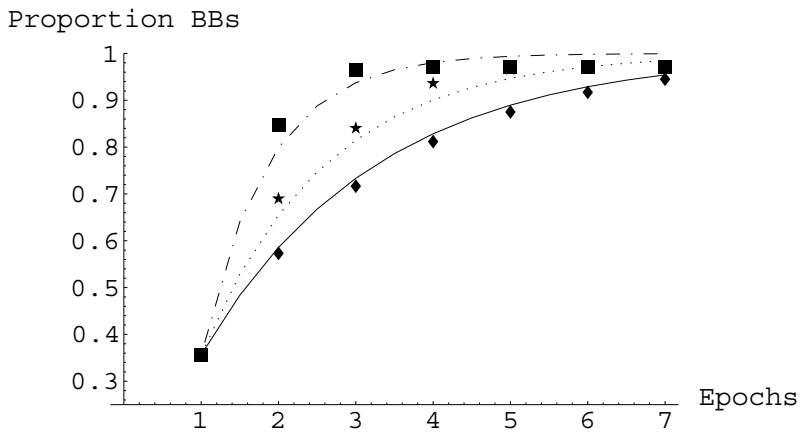


Figure 7.12: The graph compares the approximate (continuous lines) and the Markov chains models (dots) on three topologies: uni- and bi-directional rings and a fully connected topology (from right to left, respectively).

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

$$T_p = \tau \left(g n_d T_f + \delta T_c \right). \tag{7.12}$$

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

$$\delta^* = \left(\frac{g n_0 T_f}{2\tau' T_c} \right)^{2/3}, \tag{7.13}$$

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

An interesting property of the optimal degree is that it depends very weakly on r. The dependency on r is in the $\ln(1 - \hat{P})$ term of n_0 (see the discussion around equation 7.11). $\hat{P}$ 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 important because if δ^* does not change much as more demes are used, then the execution time (eq. 7.12) would not change much either. However, we should not dismiss the algorithm's capability to decrease the execution time. After all, δ^* depends strongly on the number of epochs τ, and both δ^* and the execution time 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:

$$\tau^* = 1 + \sqrt{\frac{g n_0 T_f}{\delta^{3/2} T_c}}. \tag{7.14}$$

The corresponding deme size may be found by substituting δ^* (or τ^*) in equation 7.11.

7.3 Parallel Demes in the Long Run

This section focuses on an important property of parallel GAs with multiple populations. Namely, that 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. This was established in chapter 6. We denote as τ_c the critical number of epochs necessary to reach a solution as good as a GA with an aggregate population. Chapter 6 also showed how to use Markov chains to calculate τ_c, but although those calculations are accurate, they are not very easy to use. This section shows how to estimate τ_c using the simple models for solution quality derived in the previous section.

Recall that the execution time of the parallel GA depends directly on the number

of epochs (equation 7.12). By definition, when τ_c epochs are used, the size of the demes is reduced to $n_d = \frac{n}{r}$, so the execution time becomes

$$T_p = \tau_c \left(\frac{gnT_f}{r} + \delta T_c \right). \tag{7.15}$$

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 not correct. 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:

$$\tau_c = \frac{r-1}{\sqrt{\delta}} + 1. \tag{7.16}$$

Substituting the estimate of τ_c into equation 7.15, making $\frac{\partial T_p}{\partial r} = 0$, and solving for r results in the optimal number of demes:

$$r^* = \sqrt{g \frac{\sqrt{\delta} - 1}{\delta}} \sqrt{\frac{nT_f}{T_c}}. \tag{7.17}$$

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 in this dissertation to estimate

the execution time of the two types of parallel GAs and decide which one to use.

7.4 Summary

This chapter extended the previous deme-sizing equations to consider configurations that are likely to be used by practitioners. The first part of the chapter described the relation between the deme size, the migration rate, and the topology's degree with the probability of success after two epochs. It showed how to find the configuration that optimizes the execution time while reaching a predetermined target quality. These calculations were also used to find an alternate expression for the optimal number of fully connected demes (which was calculated initially in chapter 5). Although this topology cannot integrate many demes, it can reduce the execution time substantially, and it may be competitive with other optimally-configured topologies.

The second part of the chapter generalized the results to multiple epochs. After multiple epochs, the topology is important because a deme receives indirect contributions from varying number of demes. However, section 7.2 showed that different topologies with the same degree reach almost identical solutions after any number of epochs. A simple approximate model was derived to explain the small differences, but most importantly, the equivalence of topologies with the same degree facilitated the derivation of a model of solution quality. The quality model was transformed into an accurate deme-sizing equation, which in turn was used to minimize the execution time.

When the topology is fixed and the algorithm is executed until all the populations converge to the same solution, the optimal number of populations was found to be $O(\sqrt{\frac{nT_f}{T_c}})$, which is the same as parallel versions of GAs with a single population. This result suggests that, regardless of their type, parallel GAs can integrate large numbers of processors and reduce significantly the execution time of many practical applications.

Chapter 8

Designing Hierarchical Parallel GAs

This dissertation has shown how to optimize master-slave and multiple-deme parallel GAs. At this point, we face a critical question: given a problem and a parallel computer, what algorithm returns the desired solution in the shortest time? A valid answer would be to find the optimal configurations of each algorithm and choose the algorithm with the highest speedup. However, the choice of algorithms is complicated further because there is a third option: master-slave and multi-deme GAs may be combined into hierarchical algorithms that have potentially better performance than either of their components.

The configuration of the hierarchical algorithms is easy to optimize if there are enough processors available. However, when there is a shortage of processors, they have to be allocated carefully between demes and slaves to obtain the fastest hierarchical GA possible. That is the main goal of this chapter. It integrates previous results into a method to determine the optimal combination of demes and slaves.

The chapter begins with a description of different types of hierarchical GAs in section 8.1. Then, section 8.2 discusses how to design optimal hierarchical GAs using the models for master-slave and multiple-deme GAs presented elsewhere in the dissertation. Section 8.3 gives a complete example that illustrates how to use the theory on a particular problem domain and hardware environment. The chapter ends with a brief summary in section 8.4.

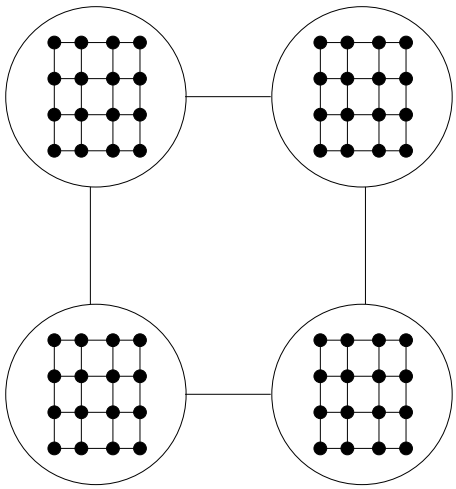


Figure 8.1: This hierarchical GA combines a multi-deme GA (at the upper level) and a fine-grained GA (at the lower level).

8.1 Hierarchical Parallel GAs

Previous chapters have shown that both master-slave and multiple-deme parallel GAs can quickly reach good solutions when they are designed properly, but even greater improvements are possible by combining the two basic forms of parallel GA into a hierarchical algorithm Grefenstette (1981) was the first to propose this. The fundamental idea is simple: use a multiple-deme algorithm where the demes themselves are some form of parallel GAs. At the upper level, the algorithm may be designed with the theory presented in the previous chapter. At the lower level, there are three choices: use a fine-grained GA, use a master-slave GA, or use a multiple-deme GA with very high migration to give the illusion of a single population. The remainder of this section examines the three types of hierarchical parallel GAs.

Figure 8.1 presents a schematic of hierarchical GA with a fine-grained GA at the lower level. A fine-grained GA consists on one population distributed on a grid; selection and recombination occur within neighborhoods defined by the structure of the grid (Manderick & Spiessens, 1989). Fine-grained GAs are also known as diffu-

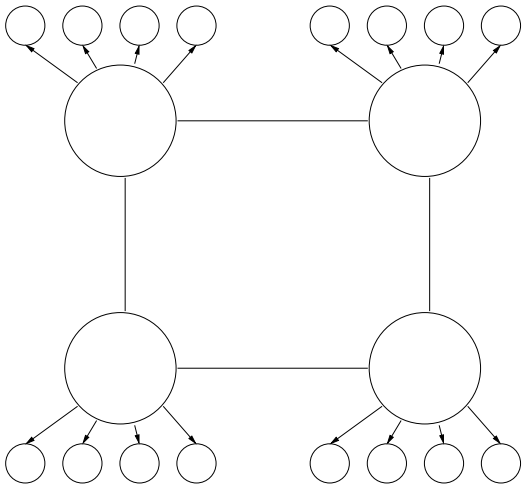


Figure 8.2: At the upper level this hybrid is a multi-deme parallel GA where each node is a master-slave GA.

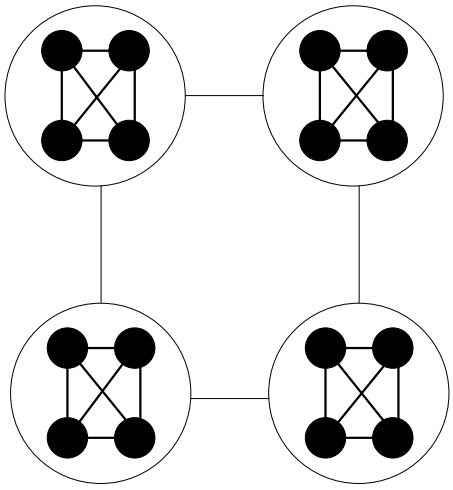


Figure 8.3: This hybrid uses multi-deme GAs at both the upper and the lower levels. At the lower level intense migration gives the illusion of a single panmictic population.

sion (Pettey, 1997) or cellular GAs (Whitley, 1993a). An example of a hybrid parallel GAs with fine-grained populations is the "mixed" parallel GA invented by Gruau (1994). In his algorithm, the population of each deme was placed on a 2-D grid, and the demes were connected as a 2-D torus. Migration occurred at regular intervals, and good results were reported for a novel neural network design and training application.

Another hybrid with a fine-grained GA at the lower level is Asparagos96 (Gorges-Schleuter, 1997). It was proposed as an extension to ASPARAGOS, one of the first fine-grained GAs ever proposed (Gorges-Schleuter, 1989; Mühlenbein, 1989). The hybrid used a ring topology to connect the demes, and the structure of the demes was also a ring. Migration was controlled by a varying rate that was very low at the start of the run, and was increased linearly—and very slowly—with the number of generations. In effect, migration is not expected to occur until the run is very advanced, somewhat similar to the migration strategy used by the multiple-deme GAs in this dissertation. After all the populations converged, Asparagos96 used the best and second best individuals of each population as the initial population of a simple GA to try to refine the solutions further.

Lin, Goodman, and Punch III (1997) also used a multi-population GA with spatially-structured subpopulations. Interestingly, they also used a ring topology to connect the subpopulations, but each deme was structured as a torus. The authors compared their hybrid against a simple GA, a fine-grained GA, two multiple-population GAs with a ring topology (one used 5 demes and variable deme sizes, the other used a constant deme size of 50 individuals and different number of demes), and another multi-deme GA with demes connected on a torus. Migration occurred at regular intervals; the best individuals of each deme were selected as migrants, and they replaced the worst individuals at the receiving deme. Chapter 5 argued that this migration scheme increases the selection pressure, and that convergence occurs faster than in a simple population. The experimental design was similar to Tanese's

where the total number of individuals is fixed, and were distributed to processors in different ways. Using a job shop scheduling problem as a benchmark, they reported that the simple GA found inferior solutions to all the other methods, and that increasing the number of demes had a larger positive effect on the solution quality than increasing the sizes of the demes or the total population size. There are no timing results for the hybrid.

Another type of hierarchical parallel GA uses a master-slave on each of the demes of a multi-population GA (see figure 8.2 for a schematic). Migration occurs as usual, and the evaluation of the individuals is handled in parallel. This approach does not introduce any new analytic problems, and it will be examined in detail in the next section. Bianchini and Brown (1993) presented an example of this method of hybridizing parallel GAs, and showed that it can find a solution of the same quality of a master-slave parallel GA or a multi-deme GA in less time.

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

The third method of hybridizing parallel GAs is to use multi-deme GAs at both the upper and lower levels (see Figure 8.3). The idea is to force panmictic mixing at the lower level by using a high migration rate and a dense topology, while a low migration rate is used at the high level (Goldberg, 1996a). We may use an algorithm similar to to the GA with the distributed panmictic population presented in chapter 3. Analytically, this hybrid would be equivalent to a multiple-population GA if the panmictic groups are considered as a single deme. This method has not been implemented.

8.2 Optimal Hierarchical Parallel GAs

This section presents a method to design an optimal hierarchical parallel GA with master-slave GAs at the lower level. Properly designed hierarchical implementations should reduce the execution time more than either of their components. In particular, the execution time of each of the upper-level demes is reduced by a factor equal to the speedup at the lower level. Therefore, the overall speedup is given by $Sp_{md} \times Sp_{ms}$, where Sp_{md} is the speedup that originates from dividing the population into multiple demes, and Sp_{ms} is the speedup from evaluating the demes using multiple slaves.

The ideal situation would be to use the optimal number of demes r^* and the optimal number of slaves $\mathcal{S}^*$. However, the number of processors available ($\mathcal{P}$) may be less than $r^* \times (\mathcal{S}^* + 1)$. If that is the case, then the processors should be allocated between demes and slaves to maximize the overall speedup. An alternative to find the optimal configuration is simply to enumerate all possible pairs, calculate the expected speedup for each, and choose the pair with the highest speedup. Although the calculations are inexpensive and the number of valid demes-slaves pairs is not too large, in fact not all of the possible pairs need to be considered.

The key to design optimal hierarchical parallel GAs is to design the upper level first and then the lower level. At the upper level, the number of demes may vary from one to the number of processors available, and since the best number of demes is unknown, this step involves enumerating all the $\mathcal{P}$ possibilities. For each deme count, the connectivity of the topology may be optimized using the theory of the previous chapter. The result is an optimal degree that can be used directly to calculate the optimal deme size using equation 7.11.

Designing the lower level of hierarchical GA is straightforward. The design consists on finding the optimal number of slaves for each deme count. Recall that the optimal number of processors in a master-slave GA is $\sqrt{n\gamma}$. In this case, n corresponds to the deme size, which was computed in the previous step for every possible

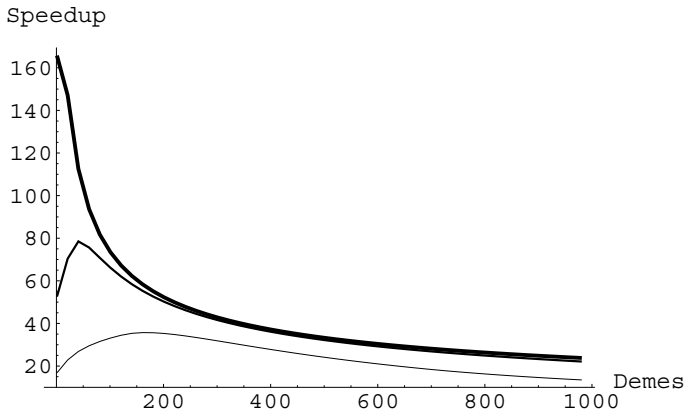


Figure 8.4: Speedups of optimally-configured hybrids of multi-deme and master-slave parallel GAs for $\gamma = T_f/T_c$ ratios of 10, 100 and 1000 (from bottom to top, respectively).

deme count. It may turn out that the optimal number of slaves is too large, and that there are not enough processors available ($\mathcal{S}^* \times r > \mathcal{P}$). In this case, the number of slaves that should be used is the largest possible, $\lfloor \mathcal{P}/r \rfloor$. Since for every deme count there is one optimal number of slaves, the number of candidate configurations that need to be evaluated is equal to the number of processors available, $\mathcal{P}$.

This method accounts the two basic forms of parallel GAs: the pure master-slave is accounted when the number of demes is one, and pure multiple-demes when $r = \mathcal{P}$. No special calculations are required to handle these cases.

Figure 8.4 shows the speedups of the optimal configurations of hierarchical GAs at all possible deme counts. The example considers a situation where there are 1000 processors available, and the three plots correspond to different values of γ (10,100,1000).

When γ is high, the master-slave GA is capable of almost linear speedups, and configurations with many slaves per deme would probably perform better than many demes with few slaves. Similarly, when communications are relatively expensive, the multiple-deme GAs have an advantage because they communicate infrequently. In this case, higher speedups are obtained using many demes and few (if any) slaves.

8.3 An Example of Optimal Design

This section presents an example that illustrates how to use the models introduced in this dissertation to obtain a desired solution in the minimum time. The test function of the example is formed by concatenating ten copies of an 8-bit fully deceptive function. This is the same test function that was used in the early chapters of the dissertation to validate the models for the bounding cases of multiple-deme GAs. The objective is to reach a solution with eight BBs ($\hat{Q} = 0.8$) in the shortest time.

To make this case study as realistically as possible, no a priori knowledge about the difficulty of the function is used. However, to facilitate comparisons with the results of other chapters, the quality of the solutions is measured as the number of partitions that converge to the correct BB.

The first step in this example is to find the hardware-dependent constants. Then, the master-slave and multiple-deme models are used together to enumerate the candidate configurations as was described in the previous section. The best configuration is chosen; and finally, experiments are performed to verify that the chosen configuration is indeed the best.

8.3.1 The Hardware

The computer used in this example is an IBM SP-2 with eight processing elements ("thin nodes" in IBM terminology). Each processing element is a Power PC 604 running at 133 Mhz with 256 Mb of memory. The nodes are connected with an IBM High Performance Switch (100 Mbits/sec) and by a 10 Mbits/sec Ethernet.

The parallel GA code was programmed in C++ and compiled with IBM's xlC compiler using the `-O2` flag to optimize the code. Communications were handled by PVM 3.4, and only the fast switch was used in the experiments.

Since the original function is very inexpensive, its execution time was artificially raised to $T_f = 1$ ms. The communication time was measured varying the size of the

packets, and for the message lengths that are likely to be used in this example the time was approximately 1 ms.

8.3.2 Finding the Best Configuration

The first step to use the theory is to calibrate the gambler's ruin model to the particular domain. A simple GA with pairwise tournament selection and one-point crossover was used to experiment. Three runs were performed with population sizes of 100, 200, and 400 individuals. The solutions found had 1, 2 and 4 correct BBs, respectively, which is far below the desired 8 BBs and indicates that much larger populations are needed.

The gambler's ruin model has two problem-dependent parameters: the order of the BBs, k, and the correct decision-making probability, p. For a given problem, however, we may group the domain-dependent components into one constant, and use the experimental results to determine its value. Specifically, the gambler's ruin problem may rewritten in the following way:

$$P_{bb} \approx 1 - \left(\frac{1-p}{p}\right)^{n/2^k} = 1 - y^n.$$

A least-squares steepest-descent method was used to fit $Q = 1 - y^n$ to the experimental data[1]. The result is that $y = 0.998786$, which is not very different from the value that would be obtained using the known theoretical values of $p = 0.5334$ and $k = 8$ for this problem ($y = 0.999022$).

The population size that a simple GA requires to find 8 BBs may now be easily determined as

$$n = \frac{\ln(1 - \frac{\hat{Q}}{m})}{\ln y} = \frac{\ln(1 - 0.8)}{\ln 0.998786} = 1325.$$

[1]Specifically, Mathematica 3.0's `NonlinearRegress` function was used as follows:
`data={{100,0.1},{200,0.2},{400,0.4}};`
`NonlinearRegress[data,1-y^n, n, y, Method->FindMinimum]`.

r	δ^*	n_d^*	$\mathcal{P}^*$	**Slaves used**
1	0	1325	36	7
2	1	519	23	3
3	2	405	20	1
4	3	345	19	1
5	4	307	18	0
6	5	279	17	0
7	6	258	16	0
8	7	241	16	0

Table 8.1: Optimal configurations for each possible deme count.

Since $T_f = T_c = 1$ms, the optimal number of processors of a master-slave parallel GA on this problem is $\sqrt{1325} = 36$, which is much greater than the eight processors available. Therefore, it is likely that very satisfactory near-linear speedups would be observed if only a simple master-slave GA is used. However, we follow the procedure outlined in the previous section to determine the optimal allocation of processors between demes and slaves.

For each deme count from one to eight, the optimal degree δ^* was computed using equation 7.13. In all cases, the optimal degree was larger than eight, and therefore the largest degree possible $(r-1)$ was used in each case.

The optimal degree $(r-1)$ was used to calculate the optimal deme size using equation 7.11. Then, using the optimal deme size, the optimal number of processors for a master-slave GA was calculated. The results are presented in table 8.1. For all deme counts, the optimal number of slaves exceeds the number of processors available, so the maximum possible number of slaves is used.

Experiments were performed with a hierarchical parallel GA using the configurations in table 8.1. The elapsed time was measured and averaged over ten trials. Each deme used pairwise tournament selection, one-point crossover with probability 1, and no mutation. The hierarchical GA found solutions with at least 8 BBs in all cases,

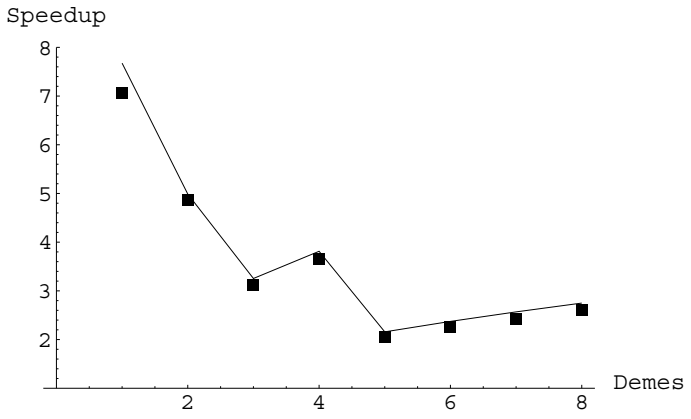


Figure 8.5: Theoretical predictions (line) and experimental results (dots) of the speedups of optimally-configured hybrids of multi-deme and master-slave parallel GAs.

except when $r = 2$ where the average solution had 7.7 BBs. The average speedup is presented in figure 8.5 along with the theoretical predictions. The experiments match the theory very well, and confirm that the best configuration for this problem has one deme and seven slaves.

If there were more processors available, the optimal configuration would be different. For example, if there were 1000 processors the optimal configuration would have 166 demes with 6 slaves per deme.

8.4 Summary

This chapter examined hierarchical combinations of multiple-deme and master-slave GAs that are capable of greater speedups than either of their parts alone. The hierarchical algorithms are composed of multiple demes, and the demes themselves are parallel GAs. The first section of the chapter identified three hierarchical algorithms that differ in the method used to parallelize the upper-level demes.

The rest of the chapter focused on hierarchical algorithms where the upper-level demes are master-slave GAs. The second section introduced a method to distribute

the available processors between upper-level demes and lower-level slaves to minimize the execution time.

Finally, section 8.3 presented a step-by-step example of the use of the theory of this dissertation to a particular problem. The example assumed minimal knowledge about the domain, and showed how only simple experiments are necessary to calibrate the theory to obtain very accurate results.

Chapter 9

Summary, Extensions, and Conclusions

This chapter first summarizes the findings and the recommendations that originate from the research contained in this dissertation. Next, it discusses possible extensions, and finally it presents the conclusions of this study.

9.1 Summary

The design of efficient and accurate parallel GAs is a complex problem. 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, making the choices difficult. The ultimate goal is to determine the configuration that reaches a solution of the required quality as fast as possible, and this dissertation made some important strides towards this goal. The investigation centered on sizing populations correctly, because the cost and the solution quality are determined mainly by the population size. The other parameters were also considered and their influence on efficiency and accuracy were quantified accurately.

The first part of this dissertation studied GAs with one population, and produced two important results. The first is an accurate model of simple GA performance based on the gambler's ruin problem. This model integrates previous knowledge of BB supply and correct decision making into an accurate predictor of solution quality. Chapter 2 presented empirical evidence that validates the accuracy of the predictions

on various settings of practical interest. The model was successfully tested with functions of varying difficulty, with badly scaled subfunctions, and with external sources of noise.

The second contribution from the first part is a lower bound on the benefits that should be expected from parallel GAs. The analysis of the simple master-slave GA in chapter 3 showed that there is an optimal number of processors $\mathcal{P}^* = \sqrt{\frac{nT_f}{T_c}}$ that minimizes the execution time, and that the maximum parallel speedup is $0.5\mathcal{P}^*$. The critical number of processors that maintain a desired efficiency was calculated, and used to determine that $\mathcal{P}_l = (\frac{nT_f}{T_c})^{1/3}$ processors maintain a near-perfect efficiency of $1 - \frac{1}{\mathcal{P}_l}$. More sophisticated parallel GAs should be more efficient and produce higher speedups or be discarded.

In addition, chapter 3 described a single-population GA that resembles a bounding case of multi-deme GAs. Although this algorithm is less efficient than the simple master-slave, it has important theoretical implications, because straightforward calculations show that it can use $O(\sqrt{\frac{nT_f}{T_c}})$ processors to reduce the execution time. This is asymptotically the same number of processors that the master-slave can use, and later chapters show that other multi-deme GAs also use this many processors.

The major part of this research focused on parallel GAs with multiple populations. These algorithms have been very popular, but the effects of their parameters on the solution quality and algorithmic efficiency are not well understood. To begin making progress in the study of multi-deme GAs, the second part of the thesis considered two bounding cases.

Chapter 4 showed how to extend the gambler's ruin model to calculate the expected quality of the best of r isolated or fully-connected demes. This chapter shows that distributing the population into multiple demes has two effects on the quality. The first is that the quality required of each deme may be reduced, because in a successful run only one of the demes has to reach the desired solution. The reduction in the target quality translates into smaller demes, which in turn represent a shorter

execution time. However, the reduction is only marginal. The second effect of using multiple demes has a greater impact on the quality and occurs when the populations communicate. After migration, the demes restart with a larger number of BBs than when they are initialized randomly, and therefore reach solutions of higher quality.

The key idea introduced in chapter 4 to predict the quality of communicating demes is to calculate how many copies of the correct BBs are present in the demes after migration. Then, the gambler's ruin model is used to predict the outcome of the second epoch. The same idea is at the core of the more refined models in the remainder of the dissertation.

The calculations in chapter 5 showed that the isolated bounding case does not result in significant performance improvements, and should be avoided in practice. On the other extreme, the size of the fully connected demes decreases substantially, thereby reducing considerably the time used by computations. However, the reduction in computations is paired with a rapidly increasing cost of communications. This tradeoff was used to find the deme size and deme count that together minimize the execution time.

Chapter 5 also discussed the controversial claims of superlinear speedups in parallel GAs. It explained that the total work performed by multiple populations may be reduced if migrants and the individuals they replace are selected according to their fitness. This additional selection pressure causes the parallel algorithm to converge in fewer generations than the serial GA. The other argument to explain superlinear speedups is that sometimes the deme sizes may be reduced by a factor greater than the number of demes and still reach the same solutions. Small demes lose diversity quickly, but migration may restore it, thereby extending the convergence time and increasing the chances that crossover mixes BBs together.

Chapter 6 used Markov chains to extend the predictions of quality to consider multiple migration epochs. The first result of the analysis is that the major improvements in quality come after the second epoch, and therefore the results from

previous chapters are relevant and particularly useful. This chapter also showed that the long-term search quality of r fully-connected demes of size n_d is equivalent to a single GA with a population of size rn_d when the migration rate is maximal. As the migration rate decreases the quality deteriorates, but even moderate migration rates are sufficient to maintain the illusion of a panmictic population. In any case, since the cost is independent of the migration rate (because migrations occur after convergence), there is no reason to avoid the highest values.

Chapter 6 also introduced the first model of arbitrary topologies, which are the topic of the third part of the dissertation. The model explicitly accounts for all the possible states of the r demes, and therefore it is very accurate, but it is also impractical for large r. Nevertheless, the model showed that the influence of the topology on the solution quality is significant and should be examined further.

Chapter 7 presented simple models that relate the deme size, the migration rate, and the degree of the topology with the quality and cost of the search. The models explain why demes with many neighbors are more likely to find the desired solution than sparsely connected demes. However, the usual tradeoff between decreasing computations and increasing communications appears, and the analysis showed how to choose the degree of the topology that minimizes the total cost. As with the bounding cases, the initial analysis considered the first two epochs, and it was later extended to arbitrary epochs.

An important observation made in chapter 7 is that the optimal degree, δ^*, does not vary considerably as the number of demes is increased. Since the execution time largely depends on the degree, this observation implies that the execution times of optimally-configured topologies do not vary much either. Therefore, any optimally-configured topology would be a good choice to reduce the execution time. Previously, the fully connected topology was optimized, and the results of chapter 7 suggest that is may be an adequate time reductor, even though it cannot connect many demes. In fact, an optimal fully connected topology uses fewer demes ($r^* = \delta^* + 1$) than any

other optimally-configured topology, and therefore it is very efficient (in terms of $\frac{Sp}{\mathcal{P}}$).

This chapter also established that when the topology is fixed and the algorithm is executed until all the demes converge to the same solution, the optimal number of populations is $O(\sqrt{\frac{nT_f}{T_c}})$, which asymptotically is the same as parallel versions of GAs with a single population. This result suggests that—regardless of whether parallel GAs use a single or multiple populations—they can integrate large numbers of processors and reduce significantly the execution time of many practical applications.

Master-slave and multi-deme GAs may be combined into a hierarchical algorithm that promises to be very efficient and scalable to large parallel architectures. Chapter 8 described different types of hierarchical parallel GAs, and examined the question of how to choose the fastest configuration. In an ideal situation, there would be enough processors available to use the optimal number of demes (r^*) and the optimal number of slaves ($\mathcal{S}$). In this case, the speedup of the best hybrid parallel GA would be the product of the speedup of the optimal master-slave GA by the speedup of the optimal multi-deme. More often, there are not enough processors to implement the best configuration, and those that are available have to be distributed carefully between demes and slaves to obtain the combination that minimizes the execution time. The chapter presented a method that integrates the theory introduced earlier to find the optimal distribution of processors.

When communications are expensive relative to computations, the best algorithm will have few slaves and many demes, because the master-slave GA communicates often. As computations become more costly, the best hierarchical configurations will likely have more slaves per deme.

The last chapter also contained a complete step-by-step example that illustrated how to apply the theory presented in this dissertation to a particular problem domain. The example discussed how to determine the domain-dependent constants present in the equations, and it used the method described earlier to find the best combination of demes and slaves.

9.2 Extensions

Although this dissertation considered many of the salient issues in the design of parallel GAs, the research can be extended in several important directions.

For instance, the second part of the dissertation studied lower and upper bounds on the topology and the migration rate. Those cases also bounded the *frequency* of migration to its lowest value: migration occurred only after convergence. That low frequency is sufficient to reach solutions of the same quality as a panmictic population, and the communications costs are kept low. However, many users of multi-deme GAs set migrations to occur every few generations or even every generation. This migration strategy introduces new options and makes the design problem even more difficult. In particular, one must decide how often to migrate, how to choose the migrants, and how to replace individuals at the receiving deme.

The method presented in this thesis can be extended to consider frequent migrations. A fundamental premise of the analysis is that the quality of the solution before migration is known. With an accurate prediction of quality as a function of the elapsed time or generations, we could use the framework of this dissertation to predict the quality after migration. Mühlenbein and Schlierkamp-Voosen (1993) developed such a model of quality vs. time for the one-max function. Their model assumes that the population is sized properly so that the GA eventually converges to the right solution, but it can be modified to consider convergence to a lesser solution. Then, the gambler's ruin model could be used to determine the required deme sizes.

Mühlenbein and Schlierkamp-Voosen's model also permits to calculate the time it takes the simple GA to converge. On the parallel case, the convergence time would be different depending on which individuals are chosen to migrate and on how the migrants replace individuals at the receiving deme. The most aggressive policy is to send copies of the best individuals and to use them to replace the worst individuals of the receiving deme, which is the most common practice. This results

in faster convergence than sending or replacing individuals at random, and it was argued before that this may be the reason why multiple populations reduce the total cost. Characterizing the takeover time of the different policies is important because if convergence occurs too fast, crossover would not have enough time to mix BBs and form good solutions.

Another direction for future research is the study of GAs with *structured* populations. These are a different form of parallel GAs that were briefly mentioned in the discussion of hierarchical parallel GAs. The basic idea is that the individuals are distributed on a lattice, and that selection and crossover occur only within a designated neighborhood. This class of parallel GAs come with its own set of parameters and choices: the size and shape of the neighborhoods, the selection and replacement of individuals, and the distribution of the population structure to multiple processors. Since this is an algorithm with one population, some questions could at least be approximated with the theory for simple GAs. Other aspects particular to these parallel GAs have been studied by others. For example, Spiessens and Manderick (1991) and Sarma and De Jong (1996) determined that varying the size and the shape of neighborhoods affects the selection pressure in a manner analogous to varying the tournament size in a simple GA. Anderson and Ferris (1990) and Schwehm (1992) examined the effect of different population structures on the quality and convergence speed of the algorithm. A better understanding of structured populations is necessary to use them with confidence as stand-alone problem solvers or as components of hierarchical parallel GAs.

9.3 Conclusions

This dissertation introduces a principled methodology towards the rational design of fast parallel GAs. Practitioners and theoreticians alike can benefit from the research reported here. 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 simple of parallel GAs. Most of the contributions of this research were made possible by focusing on one facet and assuming that the others had appropriate settings or were bounded by the cases that we examined. This work stands as a proof of principle that exact or complete models are not necessary to grasp the effects of the problem's difficulty and the parameters of parallel GAs on 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 demes or how to connect them. Instead, they can use the tools developed here to choose a configuration that yields good results fast and reliably. Some experimentation may be necessary to calibrate the models to the particular domain and hardware environment, but these experiments are cheap and easy to perform.

In particular, this dissertation made the following contributions to our understanding of parallel GAs:

Accurate population sizing for simple and parallel GAs. An adequate population size is necessary for GAs to be both effective and efficient. The gambler's ruin model for simple GAs (Harik, Cantú-Paz, Goldberg, & Miller, 1997) and its extensions to parallel GAs enable the adequately sizing of GAs to reach solutions of arbitrary quality on particular domains.

Equivalent scalability of single and multiple demes. The optimal number of processors of parallel GAs with a single population and several bounding cases of multiple communicating populations is asymptotically the same: $\mathcal{P}^* = O(\sqrt{\frac{nT_f}{T_c}})$. This means that regardless of their type (single or multiple populations) parallel GAs can integrate large numbers of processors. Although for a long time parallel GAs have been regarded as highly scalable, the closed-form expressions of $\mathcal{P}^*$ permit to quantify those claims taking into account the

specific problem domain and the hardware platform.

Lower bound on acceptable parallelism. With very simple calculations users can determine the minimum benefits they can expect from parallel implementations. An objective evaluation of the potential benefits can be very useful to make the decision of implementing a parallel GA.

Isolated demes are impractical. Users should steer away from this extreme case of parallel GAs because the savings on execution time are only marginal.

Migration greatly improves quality and efficiency. The quality improves greatly after the second epoch, and in the limit it approaches the quality that an aggregate panmictic population would reach. With higher average qualities per deme, their sizes can be reduced to obtain significant savings on the execution time.

Fully connected topologies may be adequate. Although this topology is not scalable, it reduces the execution time significantly using few demes.

Varying migration rates. In general, higher migration rates result in better solutions, but even moderately low rates are sufficient to reach solutions of the same quality as a panmictic population.

Effect of topologies. Dense topologies result in better solutions because there are more potential sources of BBs, but they are also more costly than sparse topologies. Calculations presented in this dissertation quantify the increases in quality and cost, and show how to choose a topology that minimizes the effort while still reaching the desired solution.

Optimal allocation of available resources. The theory introduced in this dissertation can be used to determine the number of demes, their size, their connec-

tivity, and the number of slaves per deme that reach the desired solution with the minimum cost.

While there is much work needed to fully comprehend all facets of parallel GAs, the results from this research constitute a collection of ideas that expand our knowledge of these algorithms. They should be incorporated into projects that use parallel GAs to improve both solution quality and algorithmic efficiency.

Bibliography

Abramowitz, M., & Stegun, I. (Eds.) (1972). *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. New York: Dover Publications.

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

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

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

Bäck, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 1 (pp. 57–62). Piscataway, NJ: IEEE Service Center.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

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

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

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

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

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

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

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

Cantú-Paz, E. (1999). Migration policies and takeover times in parallel genetic algorithms. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *GECCO-99: Proceedings of the 1999 Genetic and Evolutionary Computation Conference* (pp. 775). San Francisco, CA: Morgan Kaufmann Publishers.

Cantú-Paz, E. (1999, June). *Migration policies, selection pressure, and parallel evolutionary algorithms* (Technical Report IlliGAL Report No 99015). University of Illinois at Urbana-Champaign.

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

Cantú-Paz, E., & Goldberg, D. E. (1997). Predicting speedups of idealized bounding cases of parallel genetic algorithms. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 113–120). San Francisco: Morgan Kaufmann.

Cvetković, D., & Mühlenbein, H. (1994). *The optimal population size for uniform crossover and truncation selection* (Tech. Rep. No. GMD AS GA 94-11). Germany: German National Reseach Center for Computer Science (GMD).

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 93–108). San Mateo, CA: Morgan Kaufmann.

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

Fogarty, T. C., & Huang, R. (1991). Implementing the genetic algorithm on transputer based parallel processing systems. In Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature* (pp. 145–149). Berlin: Springer-Verlag.

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

Goldberg, D. E. (1989b). Sizing populations for serial and parallel genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 70–79). San Mateo, CA: Morgan Kaufmann. (Also TCGA Report 88004).

Goldberg, D. E. (1991). *A tutorial on genetic algorithm theory*. A presentation given at the Fourth International Conference on Genetic Algorithms, University of California at San Diego, La Jolla, CA.

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

Goldberg, D. E. (1996b). The design of innovating machines: Lessons from genetic algorithms. *Computational Methods in Applied Sciences '96*, 100–104.

Goldberg, D. E. (1998). Personal communication.

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

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

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

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, *32*(1), 10–16.

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

Goldberg, D. E., & Rudnick, M. (1991). Genetic algorithms and the variance of fitness. *Complex Systems*, *5*(3), 265–278. (Also IlliGAL Report No. 91001).

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

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

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

Grefenstette, J. J. (1995). Robot learning with parallel genetic algorithms on networked computers. In Oren, T., & Birta, L. (Eds.), *Proceedings of the 1995 Summer Computer Simulation Conference (SCSC 95)* (pp. 352–357). Ottawa: The Society for Computer Simulation.

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

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

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

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

Harik, G., Cantú-Paz, E., Goldberg, D., & Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, *7*(3).

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. In Belew, R. K., & Vose, M. D. (Eds.), *Foundations of Genetic Algorithms 4* (pp. 247–262). San Francisco: Morgan Kaufmann.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1998). The compact genetic algorithm. In of Electrical, I., & Engineers, E. (Eds.), *Proceedings of 1998 IEEE Iternational Conference on Evolutionary Computation* (pp. 523–528). Piscataway, NJ: IEEE Service Center.

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

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

Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, *2*(2), 88–105.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Isaacson, D. L., & Madsen, R. W. (1976). *Markov chains theory and applications.* New York, NY: John Wiley and Sons, Inc.

Kargupta, H. (1996). The gene expression messy genetic algorithm. In Bäck, T., Kitano, H., & Michalewicz, Z. (Eds.), *Proceedings of 1996 IEEE International Conference on Evolutionary Computation* (pp. 814–819). Piscataway, NJ: IEEE Service Center.

Lin, S.-C., Goodman, E. D., & Punch III, W. F. (1997). Investigating parallel genetic algorithms on job shop scheduling problems. In Angeline, P. J., Reynolds, R. G., McDonnell, J. R., & Eberhart, R. (Eds.), *Evolutionary Programming VI* (pp. 383–393). Berlin: Springer.

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

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

Miller, B. L. (1997). *Noise, sampling, and efficient genetic algorithms.* Doctoral dissertation, University of Illinois at Urbana-Champaign. Also available as IlliGAL tech report No. 97001.

Miller, B. L., & Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131.

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

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Voigt, H.-M., Ebeling, W., Rechenberg, I., & Schwefel, H.-P. (Eds.), *Parallel Problem Solving from Nature, PPSN IV* (pp. 178–187). Berlin: Springer-Verlag.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1994). The science of breeding and its application to the breeder genetic algorithm (BGA). *Evolutionary Computation*, *1*(4), 335–360.

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

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *GECCO-99: Proceedings of the 1999 Genetic and Evolutionary Computation Conference* (pp. 525–532). San Francisco, CA: Morgan Kaufmann Publishers.

Pettey, C. C. (1997). Population structures: diffusion (cellular) models. In Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation* (pp. C6.4:1–C6.4:6). Bristol and New York: Institute of Physics Publishing and Oxford University Press.

Punch, W. F. (1998). How effective are multiple populations in genetic programming. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., & Riolo, R. L. (Eds.), *Genetic Programming 98, Proceedings of the Third Annual Conference* (pp. 308–313). San Francisco: Morgan Kaufmann Publishers.

Sarma, J., & De Jong, K. (1996). An analysis of the effects of neighborhood size and shape on local selection algorithms. In Voigt, H.-M., Ebeling, W., Rechenberg, I., & Schwefel, H.-P. (Eds.), *Parallel Problem Solving from Nature, PPSN IV* (pp. 236–244). Berlin: Springer-Verlag.

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

Spiessens, P., & Manderick, B. (1991). A massively parallel genetic algorithm: Implementation and first analysis. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 279–286). San Mateo, CA: Morgan Kaufmann.

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

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

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

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.

Valenzuela-Rendón, M. (1989). *Two analysis tools to describe the operation of classifier systems*. Doctoral dissertation, University of Alabama, Tuscaloosa. Also available as TCGA Report No. 89005.

Whitley, D. (1993a). Cellular genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 658). San Mateo, CA: Morgan Kaufmann.

Whitley, D. (1993b). An executable model of a simple genetic algorithm. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 45–62). San Mateo, CA: Morgan Kaufmann.

Whitley, D., Rana, S., & Heckendorn, R. B. (1999). Exploiting separability in search: The island model genetic algorithm. *Journal of Computing and Information Technology*. forthcoming.

Zeigler, B. P., & Kim, J. (1993). Asynchronous genetic algorithms on parallel computers. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 660). San Mateo, CA: Morgan Kaufmann.

Vita

ERICK CANTÚ-PAZ

Home Address

803 West Illinois St.

Urbana, IL 61801

Phone: (217) 384-9866

cantupaz@illigal.ge.uiuc.edu

http://www-illigal.ge.uiuc.edu/∼cantupaz

Work Address

104 South Mathews Ave.

Urbana, IL 61801

Phone: (217) 333-2346

Fax: (217) 244-5705

Education

1994-1999 Ph.D. in Computer Science

University of Illinois at Urbana-Champaign

1989-1994 B.S. in Computer Engineering

Instituto Tecnológico Autónomo de México

Research Interests

Genetic and Evolutionary Computation, Machine Learning, Data Mining, Distributed Artificial Intelligence, Stochastic Optimization.

Experience

August 1999–present: Post Doctoral Research Staff Member

Lawrence Livermore National Laboratory.

August 1994–July 1999: Graduate Fellow

University of Illinois at Urbana-Champaign.

Doctoral research focused on the study and design of reliable, efficient and scalable parallel genetic algorithms. Derived accurate models of expected solution quality, and analyzed how communication between populations affects the search quality and the algorithmic cost. These findings were used to optimize three types of parallel genetic algorithms to reach the desired solutions using minimum time. Advisor: David E. Goldberg.

October 1998–July 1999: Graduate Research Assistant

Beckman Institute, University of Illinois at Urbana-Champaign.

Enhanced the effectiveness and efficiency of the genetic algorithm engine of a decision support tool that generates military maneuver plans.

August 1997–July 1999: Laboratory Manager

Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Oversaw the daily operations of the lab, both physically and on the Internet. Supervised four undergraduate assistants and a system administrator. Developed and executed hardware and software upgrade plans. Interacted with visitors and industrial contacts.

January 1997–May 1998: Graduate Teaching Assistant

Department of Computer Science, University of Illinois at Urbana-Champaign.

Taught Introduction to Computing with Applications to Business and Commerce. Conducted laboratory sessions and lectured occasionally. Acted as co-head TA for one year, organizing the efforts of twelve graduate assistants, creating instructional material, and coordinating with other campus units.

January 1994–July 1994: Lecturer

Instituto Tecnológico Autónomo de México, Mexico City.

Developed the syllabus and taught a parallel programming laboratory for Computer Engineering undergraduates. Also taught Introduction to Computer Science to non-engineering majors.

January 1994–May 1994: Lecturer

Universidad Anáhuac del Sur, Mexico City.

Taught a C programming class to engineering undergraduates.

Awards

Fulbright Graduate Fellowship, U.S. Information Agency, 1994-1998.

Graduate Scholarship, Consejo Nacional de Ciencia y Tecnología (CONACyT), México, 1994-1998.

Honorific Mention, Instituto Tecnológico Autónomo de México, 1994.

Academic Achievement Recognition, Mexican Society of Mechanical and Electrical Engineers, 1994.

Technical Skills

Skills: Genetic and Evolutionary Computation, Machine Learning, Artificial Intelligence, Parallel Computation, Object Oriented Design and Programming

Languages: C++, C, Java, HTML, Visual Basic, Pascal, Lisp, PVM.

Environments: Unix (Linux, AIX, Irix), Microsoft Windows 95 and NT.

Machines: PCs, SGIs, IBM-RS6000, Sun Sparcstations, IBM-SP2, CM-5.

Professional Activities

Editorial Board Member, Evolutionary Computation Book Series, Instituto Politécnico Nacional, México.

Electronic Media Chair, Parallel GAs Session Chair, Member Program Committee, Genetic and Evolutionary Computation Conference, July, 1999.

Evolutionary Computation and Parallel Processing Workshop Co-Chair, July, 1999

Co-instructor, Short Course on Genetic Algorithms, February 9, 1999.

Co-instructor, Short Course on Genetic Algorithms, December 1-4, 1998.

Session chair, Genetic Programming Conference, July, 1998.

Reviewed papers for:

Computer Methods in Applied Mechanics and Engineering

Evolutionary Computation

IEEE Transactions on Systems, Man, and Cybernetics

Journal of Applied Intelligence

Journal of Parallel and Distributed Computing

Member, ACM, 1994–present.

Publications

Cantú-Paz, E. and Goldberg, D.E. (In press) On the scalability of parallel genetic algorithms. *Evolutionary Computation.*

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

Harik, G., Cantú-Paz, E., Goldberg, D.E., and Miller, B. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, *7*(3). (extended version of Harik et al, 1997)

Cantú-Paz, E. (1999). Migration policies and takeover times in genetic algorithms. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference.* (p. 775). San Francisco, CA: Morgan Kaufmann.

Cantú-Paz, E. (1999). Topologies, migration rates,and multi-population parallel genetic algorithms. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference.* (pp. 91–98). San Francisco, CA: Morgan Kaufmann.

Pelikan, M., Goldberg, D.E., and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference.* (pp. 525–532). San Francisco, CA: Morgan Kaufmann.

Cantú-Paz, E. (1999). Implementing fast and reliable parallel genetic algorithms. In Chalmers, L. (Ed.), *Handbook of Practical Genetic Algorithms.* Volume III. pp. 65–84. CRC Press.

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

Cantú-Paz, E. (1998). Using Markov chains to analyze a bounding case of parallel genetic algorithms. In Koza, J., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo,

M., Fogel, D., Garzon, M., Goldberg, D. E., Iba, H. & Riolo, R. (Eds.), *Genetic Programming: Proceedings of the Third Annual Conference.* (pp. 456-462). San Francisco, CA: Morgan Kaufmann.

Cantú-Paz, E. (1998). Designing efficient master-slave parallel genetic algorithms. In Koza, J., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Goldberg, D. E., Iba, H. & Riolo, R. (Eds.), *Genetic Programming: Proceedings of the Third Annual Conference.* (pp. 455). San Francisco, CA: Morgan Kaufmann.

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

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

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

Cantú-Paz, E. and Mejía Olvera, M. (1995). Algoritmos genéticos paralelos. *Soluciones Avanzadas.* 3 (17). 12–19. (In Spanish)

Cantú-Paz, E. and Mejía Olvera, M. (1994). Experimental results on distributed genetic algorithms. In *Proceedings of the Second International Symposium on*

Applied Corporate Computing. (pp. 99–107). Monterrey, México.

Mejía Olvera, M. and Cantú-Paz, E. (1994) DGENESIS—Software para la ejecución de algoritmos genéticos distribuidos. In *Memorias de la XX Conferencia Latinoamericana de Informática* (pp. 935-946). Atizapán de Zaragoza, México. (In Spanish)

Technical Reports

These manuscripts have been submitted to journals or conferences. Most are available as technical reports of the Illinois Genetic Algorithms Laboratory (`http://www-illigal.ge.uiuc.edu`).

Cantú-Paz, E. (1999). Migration policies, selection pressure, and parallel evolutionary algorithms. Available as IlliGAL technical report 99015.

Cantú-Paz, E. and Goldberg, D.E. (1999). Parallel genetic algorithms with distributed panmictic populations. Available as IlliGAL technical report 99006.

Pelikan, M., Goldberg, D.E., and Cantú-Paz, E. (1998). Linkage problem, distribution estimation, and Bayesian networks. Available as IlliGAL technical report 98013.

Cantú-Paz, E. (1998). A Markov chain analysis of parallel genetic algorithms with arbitrary topologies and migration rates. Available as IlliGAL technical report 98010.

References available upon request