Efficient Genetic Algorithms Using Discretization Scheduling

Laura A. McLay lalbert@uiuc.edu
Department of Mechanical and Industrial Engineering, University of Illinois, Urbana, Illinois 61801, USA

David E. Goldberg deg@uiuc.edu
Department of General Engineering, University of Illinois, Urbana, Illinois 61801, USA

Abstract

In many applications of genetic algorithms, there is a tradeoff between speed and accuracy in fitness evaluations when evaluations use numerical methods with varying discretization. In these types of applications, the cost and accuracy vary from discretization errors when implicit or explicit quadrature is used to estimate the function evaluations. This paper examines *discretization scheduling*, or how to vary the discretization within the genetic algorithm in order to use the least amount of computation time for a solution of a desired quality. The effectiveness of discretization scheduling can be determined by comparing its computation time to the computation time of a GA using a constant discretization. There are three ingredients for the discretization scheduling: population sizing, estimated time for each function evaluation and predicted convergence time analysis. Idealized one- and two-dimensional experiments and an inverse groundwater application illustrate the computational savings to be achieved from using discretization scheduling.

Keywords

Genetic algorithms, efficiency scheduling, solution quality, speedup, convergence time, population sizing, discretization

1 Introduction

In recent years, genetic algorithms (GAs) have solved an increasing number of complex real-world problems in optimization, search, scheduling, and machine learning. One obstacle for the use of GAs in industrial applications is the high evaluation expense. Since GAs typically require hundreds or thousands of function evaluations, and industrial applications may require anywhere from minutes to days for a single evaluation, a GA may require an unacceptably long time to complete. Some of these applications with expensive function evaluations include computational fluid dynamics, structural optimization, and environmental applications. In many cases, coarse-grained discretization can be used to decrease the computation time by introducing additional error in the fitness evaluations. This paper focuses on how these types of problems can run effectively and accurately while dealing with considerable amounts of error in the evaluation.

The ultimate goal of this research effort is to establish efficient guidelines for GAs that use a finite discretization to approximate a continuous system. To achieve this goal, an understanding of the effects of evaluation error on GA performance is obtained, and a model is created that will both predict the accuracy and computational requirements in an environment with evaluation error present. Although the specific situations are somewhat idealized, the general idea applies to more complex evaluations resulting from implicit or explicit quadrature in finite elements, finite differences, or other techniques used to approximate differential and integral equations.

This paper considers the following question: How can the discretization be scheduled in order for the least amount of computation time to be spent toward finding a solution of a given quality? The central focus is threefold: (1) to develop a framework to describe and analyze the effects of error of fitness evaluations in GAs, (2) to predict computation time needed to achieve a certain level of accuracy for a given amount of error, and (3) to determine the optimal schedule of discretization to most efficiently use computation time.

This paper addresses these issues in a genetic algorithm that uses implicit or explicit quadrature to estimate the fitness. Section 2 defines the error terminology used in this paper and provides background on related work. Section 3 provides an empirical analysis of a fixed grid discretization scheme as well as the motivation for discretization scheduling. Section 4 contains the components for discretization scheduling, including an error framework, convergence time models and population sizing, and introduces discretization scheduling and its components. Section 5 discusses naive and efficient discretizations and speedup, for experiments in one-dimension as well as idealized one-dimensional experiments. Section 6 generalizes the results from Section 5 into multiple dimensions and contains idealized two-dimensional experiments. Section 7 contains the inverse groundwater problem and its analysis. Finally, in Section 8, this paper is concluded.

2 Background

This section covers two topics. First, error analysis is characterized by its components and each component is defined. Next, a literature review of efficient GAs summarizes research done in this field.

2.1 Error Characterization

Evaluation error can be quantified into its components by considering it as error from a biased estimator. Error of a biased estimator is composed of two components: bias and variance. Error can be quantified as

$$E = B^2 + \sigma_E^2$$

where B is the bias and σ_E^2 is the variance.

If the evaluation error is due to randomness or variance, then the error is unbiased. This type of error is called *variance error*, often called *noise* in the GA literature. In other cases, the evaluation error cannot be averaged away, and a good part of the error is due to bias. This type of error is called *bias error*.

When a finite discretization of a continuous system is used to estimate fitness evaluations, a truly accurate measurement of the fitness is not possible. When functions that use the finite element method, finite differencing, or numerical integration, for example, the error as a result of the inaccurate fitness evaluation may have both bias and variance components.

This paper considers numerically integrated fitness evaluations which may have a bias component present and whose estimated fitness, f', can be written as

$$f' = f + evaluation\ error,$$

where f is the accurate fitness. The error in this case is distributed $N(B, \sigma_{E,t}^2)$, and both bias and variance components may be significant in this system. If the variance component is insignificant compared to the bias component (i.e., the error is roughly distributed $N(B, 0)$), then the error can be referred to as *bias error*, and the objective function will simply be shifted by B.

In the following section, the bias component is isolated and empirically analyzed. The results of these experiments serve as the motivation for discretization scheduling, where the variance component plays a larger role.

A brief overview of error in numerical integration and finite difference methods is given. The error estimates, as well as the computational requirements, from using these methods is critical for predicting the computation and convergence times of GAs. It is assumed that numerical integration scheme used to estimate the fitness can be written of the form:

$$I(f) = \int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

where the integral of the function f taken at the points x_i is taken from a to b, n is the total number of grid points, and w_i are the weights as determined by the numerical integration scheme. The estimated error in the integration scheme is accurate to a certain order depending on h, the size of the discretization where $h = \frac{b-a}{n}$. For the left-endpoint rule, the integral is O(h) accurate; for the midpoint and trapezoidal rules, the integral is O(h^2) accurate; and for Simpson's rule, the integral is O(h^4) accurate. In addition to the accuracy, there are different computational requirements to implement different numerical integration schemes. It is assumed that the time to estimate an integral is dependent only on the number of grid points, n.

Finite differencing can be written in ways that are first or second order accurate. Forward and backward difference formulas are O(h) accurate while centered difference formulas are O(h^2) accurate. Centered differencing is used for the calculations in groundwater experiments in Section 7. The formula is given by

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

where h is the size of the discretizations and f' is the derivative with respect to x. As for numerical integration, it is assumed that the time to implement finite differencing is proportional to the number of grid points, n.

2.2 Background of Efficient GAs

In this section, the background of GAs with evaluation error and efficient GAs are discussed. The first part discusses how fitness functions with variance error affect GAs. The second part concerns efficient industrial applications using GAs, whose fitness evaluations may be expensive.

Early research considered the error-time tradeoff for sampling problems, in which an optimal sample size or sampling strategy was sought. The error introduced to these problems can be categorized as variance error because the bias is approximately zero. Moreover, early studies on the effects of evaluation error on bounded GA performance were purely empirical. Grefenstette and Fitzpatrick (1985) empirically found that using smaller sample sizes, which resulted in noisier fitness evaluations, increased the performance of a GA because the total number of fitness evaluations made in a given time was larger. Fitzpatrick and Grefenstette (1988) found that an optimal sampling size exists for a given ratio. When the GA overhead was negligible, performance was best with larger population sizes and smaller sample sizes, maximizing the total number of fitness evaluations in any given time. When the overhead per individual per generation was significant, the population and sample sizes yielded the best results when somewhat larger sample sizes were used when the overhead was larger. Aizawa and Wah (1993, 1994) developed methods to dynamically change the number of samples and the time spent on each generations while keeping the overall population size constant. Since less accuracy is needed for evaluations in the early generations, smaller sample sizes are better early on and larger sample sizes are better in the later generations.

Schaefer (1987) and Schraudolph and Belew (1992) empirically considered exponentially (non-uniformly) scaled building blocks when sampling is used to estimate fitness values. They both developed zoom operators that, once a bit has converged to a certain proportion, would restrict subsequent search to a target interval. Schraudolph and Belew re-initialized the least significant bit after a zoom operation to dissuade the least significant bits from deviating from their randomized states due to genetic hitchhiking, which sometimes caused premature convergence of the GA.

Miller and Goldberg (1996) provided the theory for optimal sampling for problems with internal error present. They found that a lower bound to the sampling existed; if fewer samples than this lower bound were used, the GA was likely to converge to a sub-optimal solution. Miller (1997) explored the optimization of performance and sampling techniques of GAs in inaccurate function evaluations in his dissertation, and he explained how internal error affects the population sizing as well as the performance of a GA and its computational requirements.

Other types of research regarding efficient GAs considered evaluation error that may have a bias component present. This evaluation error is either explicitly considered or indirectly considered by examining how to efficiently handle expensive fitness evaluations that may be present in industrial applications. These GAs are run either serially or in parallel, but all decrease the cost of each evaluation or the average evaluation cost.

Some research efforts examine using approximate models to decrease evaluation expense. Ratle (1998) found that the convergence rate of real-coded evolutionary algorithms could be improved by creating an approximate model of the fitness landscape with statistical techniques, decreasing the evaluation cost and total required fitness evaluations by minimizing the complexity of the fitness function. El-Beltagy, Nair, and Keane (1999) coupled metamodeling techniques to evolutionary algorithms to reduce computation costs. However, both of these techniques assumed that the approximated models lead the GA to the correct optima. Jin, Olhofer, and Sendhoff (2000) addressed this concern by empirically examining the convergence property of an evolution strategy with neural network based evaluations. They found that false minima with the approximated fitness can be eliminated by using regularization techniques. Later, they examined evolution control to ensure that evolutionary algorithms using approximate fitness values converge correctly to the actual global optima (2001).

Others used additional heuristics to improve GA performance for specific types of problems. Le Riche and Haftka (1993) introduced a new genetic operator—permutation—that reduces the cost of genetic search by up to a factor of 2 for some problems. Permuting parts of bit strings instead of using mutation was particularly effective for problems considering buckling optimization. Von Wolfersdorf et al. (1997) considered shape optimization for convective cooling channels in a two-dimensional region in which a large part of the evaluation time was consumed by generating new finite element models. Kogiso et al. (1993) improved the efficiency of fitness functions using the finite element method by storing the previous solutions in a binary tree in order to inexpensively retrieve previous fitness values, and by using a local improvement scheme around nominal designs to approximate fitness values.

The injection island genetic algorithm (iiGA) refined the model in a stepwise process by running multiple grids for large engineering applications in parallel. This method was first developed by Lin et al. (1994) and Punch, et al. (1994) and has been used by others for shape optimization with finite elements (Parmee et al., 1997; Eby et al., 1997; Fernández et al., 2000) . For these applications, each fitness evaluation is very costly and traditionally, GAs used coarse grid spacing to solve these types of problems. The authors intuitively understood that some good building blocks could be generated inexpensively by using a coarse grid spacing, so they ran a number of different resolution grids in parallel, each of which was run on a different set of processors or *islands*, each with its own subpopulation. They then injected the best individuals into the higher resolution grids. They found that the subpopulations explored different areas of the search space simultaneously and considered multiple encodings, which could overcome difficulties from Hamming distances when the string was represented in binary. They found that the best individuals from the low resolution grid provided good building blocks for the higher resolution grids at a cheaper cost than the high resolution grids for the same building blocks.

Albert and Goldberg (2000) empirically examined the tradeoffs between more and less accurate models that use numerical integration. They found that for a bounded computation time, an optimal number of grid points exists that can maximize solution quality. Albert and Goldberg (2001) examined evaluation relaxation under integrated fitness functions. They varied the discretization by using fewer grid points in the early generations and exponentially increasing the number of grid points in order to more efficiently use computation time. Although more function evaluations are needed for the efficient method compared to when the number of grid points is constant for the entire GA, they achieved a significant computational speedup from their method. Albert (2001) presented a framework for varying the discretization in GAs with fitness evaluations that use numerical methods.

3 Constant Grid Analysis

This section reviews the implications of using a constant grid size in a genetic algorithm, and also explains the motivation for discretization scheduling. This section begins with a qualitative description of the error-accuracy tradeoff, and then continues to to show experiments that verify the predictions.

For many fitness functions, a tradeoff exists between increased accuracy for each evaluation and running more error-prone evaluations for any given amount of computation time. In many cases, error itself is unavoidable either because a perfectly accurate function does not exist or may be infeasible in terms of required computation time. Historically, GAs using quadrature in the fitness evaluations typically consider a constant grid size for the duration of the GA, and optimizing performance in a single processor by scheduling the discretization has not yet been addressed.

Much of the previous, theoretical research on the effects of error on GA performance has considered only variance error where sampling is used to relax the fitness evaluations. The tradeoff between the number of samples and the solution quality can be extended to other numerical methods using discretization commonly used in engineering disciplines.

Albert and Goldberg (2000) analyze this tradeoff in the context of one-dimensional numerical integration. They predict that the optimal grid spacing is the smallest grid spacing such that the GA has time to complete. In other words, the grid spacing that has the expected computation time which matches the given amount of available computation time will yield the solution with the least amount of error. When the grid spacing is finer than the optimum, each evaluation is accurate but so expensive that the GA does not have time to finish in the amount of time given. In this case, the GA is halted prematurely. When the grid spacing is coarser than the optimum, the solution quality is limited by the coarseness of the grid. In this case, the GA finishes but each evaluation is so inaccurate that the error introduced prevents the GA from finding a higher quality solution.

The experiments in this research effort isolate the bias component of the error to see how it is handled within the GA. The problem considers a simply supported beam with an applied vertical load. The beam has a constant width and is divided into a possible number of equally-spaced intervals, each of which can take a value of 8 discrete, equally spaced heights. Thus, the grid spacing $h = \frac{L}{d}$, where L is the beam length. The objective is to minimize the average sum-squared difference between the stress in each interval of the beam and the allowable stress:

$$f = \frac{1}{d^2} \sum_{i}^{d} (\sigma_A - \sigma_i)^2$$

where f is the fitness value, d is the total number of intervals in the beam, σ_A is the allowable stress and σ_i is the stress at the ith interval of the beam.

Integration is used to estimate the fitness evaluations, and since the evaluations are deterministic, the error introduced from the integration is purely biased. For a fixed grid spacing, the introduction of bias error does not increase the selection intensity or increase the convergence time. The midpoint rule was used to approximate the moment used for the fitness evaluation, and it is used in conjunction with the depth at each interval to determine the fitness. This rule is $O(h^2)$—halving the size of the interval size, h, results in a quarter of the error.

The GA ran for the fixed amount of computation time on a Pentium III with a 550 MHz processor and 64 MB of RAM. The computation time was limited to 2×10^6 total floating point operations (labeled 2.0 Mflops). The problem was encoded as a simple genetic algorithm using binary tournament selection, one-point crossover, and probability of mutation as 0.01 using Matlab. The results were averaged over 10 trials for various values of d. The expected computation time needed by the GA was estimated for different values of d based on the population size and convergence time necessitated by any such number of intervals, d.

It is predicted that the optimal grid spacing (and, hence, the number of intervals in the problem) occurs when the expected computation time required by the GA with this grid spacing is the same as the fixed amount of computation time. This happens when d is approximately 7.7 intervals. The results are shown in Figure 1. As expected, there is an optimal number of intervals when the GA is run for a fixed amount of computation time. This optimal number of intervals is approximately the same as the predicted value. To the left of the optimum, there are too few intervals to accurately estimate the fitness evaluations. To the right of the optimum, there are too many intervals—in other words, the grid size is too small—and the GA cannot finish in the given computation time. The optimal value is consistent with the predicted optimal value of 7.7.

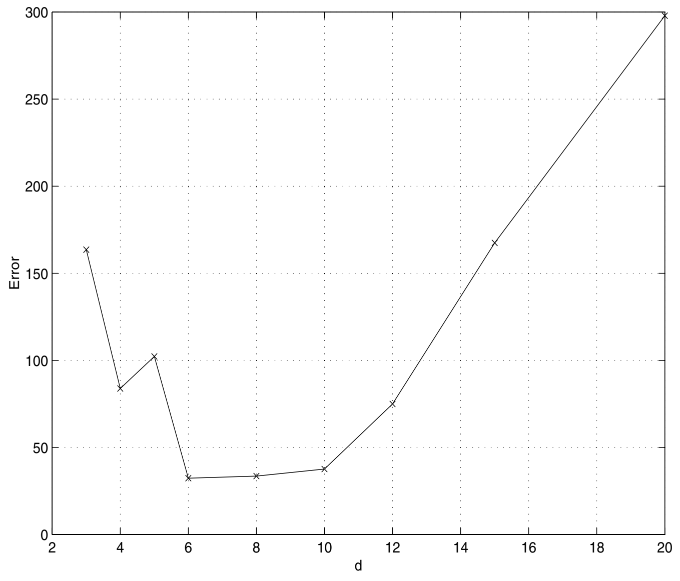

Figure 1: The error-time tradeoff for constant grid experiments.

3.1 Motivation for Discretization Scheduling

In these experiments, the solution quality was maximized for a given amount of time. This implies that for any value of time that is available, there is a limit to the value of the solution quality available. If this value of the solution quality is not meaningful for a particular problem, the GA practitioner may desire to minimize the computation to acquire a given level of the solution quality. This may be more practical in the sense that high performance computing time is often limited or can be purchased, so using the least amount of computation time is desirable.

In this context, discretization scheduling is defined as using more than one grid size within the GA. It is predicted that varying the grid size can improve the solution quality faster by starting with a coarse grid and switching to finer grids. In the following sections, how discretization scheduling can optimize computation time for a desired level of solution quality is discussed.

4 Ingredients for Discretization Scheduling

This section focuses on discretization scheduling and its components. We begin with the theoretical basis behind discretization scheduling, including an error framework and apparent variance error. The rest of the section focuses on the effect of error on solution quality, selection intensity, convergence time and population sizing.

Evaluation error introduced by numerical methods may have both bias and error components. Purely biased evaluation error does not decrease the selection intensity, increase the convergence time, or require a larger population size, which the experiments in Section 3 illustrate. This section assumes that the bias component of the evaluation error has a negligible effect on the selection intensity, convergence time, and population size as compared to the variance component.

4.1 Discretization Scheduling

This section focuses on an error framework that models GAs that may have bias error in their fitness evaluations. It is inspired by error analysis typical in numerical solutions or differential equations (Dahlquist, 1974). The choice of interpolation functions produce two possible results from evaluation error: a shift in the *amplitude* or a shift in the *phase*. A difference in amplitude, denoted by γ, may lead the GA to the correct optimum but the fitness value differs by a quantity γ. A phase shift, denoted by δ, is measured as the largest difference between the actual maximum and the closest peak within γ. In this case, the fitness function values returned by the GA may be roughly correct, but they are skewed over the search space by some distance δ.

The phase shift is desired to be smaller than the discretization size. If this is the case, the error is said to be *usable*, and the GA is likely to converge to the optimal solution. If the phase shift is larger than the discretization size, then building blocks may be lost, and the GA converges to a suboptimal solution. For a very fine-grained discretization, the phase shift is typically less than *h*, but for a coarse-grained discretization, the phase shift is likely to be larger than *h*. In the latter case, if the discretization changes within the GA, the phase shift must be less than the discretization size for every grid spacing.

Figure 2 illustrates these concepts for an arbitrary objective function whose objective is to maximize the fitness. In this figure, *h* is simply the size of each discretization and γ is the difference between the actual and approximate maxima. In this case, δ is the difference between the position of the actual maximum and the position of the furthest endpoint of the discretization, because the segment including the maximum of the approximate function also includes the maximum of the actual function. In other words, the discretization is within the usable level needed by the GA to converge to the optimal solution. Both the amplitude and phase shifts are expected to approach zero as the grid spacing becomes smaller. It is expected that the fitness approximation approaches the true fitness as the discretization becomes finer.

As seen in Section 3, using a constant grid with a deterministic fitness function adds bias error to the evaluations. That is, for any value of the variables, the same fitness value will be assigned at any time during the GA. When the discretization is scheduled—more than one grid spacing is used by the GA—*apparent variance error* is introduced into the GA. That is, although the evaluations are determinate, each time the grid changes, the fitness evaluation for a given set of variables also changes. When the grid changes often, the variance term in the error can become significant. In practice, this means that the GA will experience duration elongation and will necessitate a larger population.

The ratio of the error variance to the fitness variance, *r*, at any time during the GA is assumed to be constant since both depend on the accuracy of the numerical scheme used as well as the distribution of individuals in the fitness landscape.

$$r = \frac{\sigma^2_{E,t}}{\sigma^2_{F,t}} \tag{1}$$

In addition, both variances decrease with time and are approximately zero at convergence. For the computational experiments in the following sections, it is empirically shown that *r* is approximately constant during the GA when an efficient discretization is used.

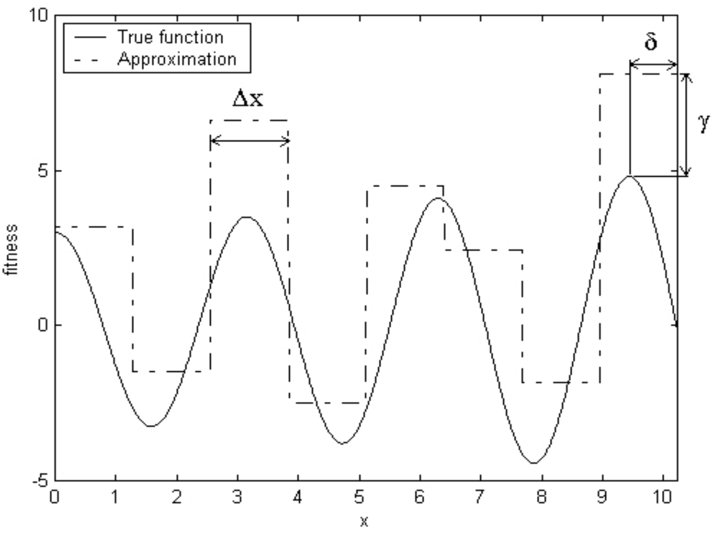

Figure 2: Schematic of amplitude and phase shifts.

4.2 Selection Intensity

The *selection intensity, I,* is the measure of the magnitude of the selection pressure by the selection scheme used by the GA. Mühlenbein and Schlierkamp-Voosen (1993) show that when the selection intensity is known and the population is normally distributed $N(\mu_t, \sigma^2_{F,t})$ at generation *t*, the mean of the following generation can be determined by

$$\mu_{t+1} - \mu_t = I\sigma_{F,t}. \tag{2}$$

With approximate fitness values, GAs have lower selection intensities and slower convergence times than if accurate fitness evaluations were available (Miller and Goldberg, 1996a; Miller and Goldberg, 1996b). Since the selection process itself is based on the fitness values, inaccurate fitness values will make the selection operator itself look error-prone. Evaluation error decreases the selection intensity, and the factor that it is decreased depends on the error variance as compared to the fitness variance. Miller (1997) shows that (2) can be rewritten for functions in the presence of variance error as

$$\mu_{t+1} - \mu_t = I\frac{\sigma^2_{F,t}}{\sqrt{\sigma^2_{F,t} + \sigma^2_{E,t}}} \tag{3}$$

for ordinal-based selection schemes, such as tournament selection where $\sigma_{F,t}$ is the fitness variance and $\sigma_{E,t}$ is the error variance at time t. Miller finds that (2) remains unchanged for proportionate selection schemes.

4.3 Convergence and Drift Models

The convergence time equations for exponentially-scaled building blocks are introduced, and the convergence time for a special case of a sum of identical subfunctions is derived. The convergence times are based on selection intensity adapted from population genetics (Kimura, 1964). The specific convergence times for both accurate functions and functions with evaluation error present are derived in Sections 5 and 6 for ordinal-based selection schemes.

Binary integer subcodes experience domino convergence, which converge starting with the most salient building block down to the least salient building block. Since binary integer subcodes have exponentially-scaled bits, it is assumed that other exponentially-scaled codes also converge in this way. In essence, this indicates that not all parts of the problem are solved at the same time, and the most salient building blocks are discovered by the GA first.

Thierens et al. (1998) created the λ model which marks the dividing line between the converged bits and the bits that are unaffected by the GA, in which λ, the number of converged building blocks, can vary from 1 to l, the string length. They show that for selection operators with constant selection intensity, such as tournament selection, the population converges in O(l) time where l is the string length. The convergence time (measured in the number of generations) for binary integer subcodes can be expressed as

$$t_{conv} = \frac{-\ln 2}{\ln\left[1 - \frac{I}{\sqrt{3}}\right]}\lambda = c_t \cdot k \tag{4}$$

where I is the selection intensity, k is the number of building blocks, and c_t is the time to converge to each successive building block. This equation implies that it takes the same amount of time to converge to each building block, regardless of its fitness contribution. When the whole string converges, λ is equal to the string length l.

Because the least salient building blocks do not experience any selection pressure for a number of generations, it is possible that they will experience genetic drift and will converge randomly to either 0 or 1. The expected time for a bit to experience drift is proportional to the population size (Goldberg and Segrest, 1987),

$$t_{drift} \approx 1.4N, \tag{5}$$

where N is the population size.

The convergence time of a special case of fitness functions is derived to be used in the idealized experiments in Section 5. When a fitness function is composed of the sum of m identical subfunctions, it has the form

$$f = \sum_{i=1}^{m} g(x_i),$$

where f is the fitness function, g is the subfunction, and x_i is the ith variable. Each subfunction contributes identically to the fitness and the variance of each subfunction is identical. Although this fitness function is of an idealized form, this may be valid as an approximation to real-world fitness functions either directly or indirectly, when weights are used to normalize the contribution of each of the subfunctions to the fitness.

In the case of a single subfunction, the increase in the mean fitness of the population from one generation is shown by (2). If f is composed of m subfunctions, each of which is added to the others, the difference in the means of two populations increases by a factor of m, and the standard deviation increases by a factor of $\sqrt{m}$. This model can be rewritten for this special case as:

$$m(\mu_{t+1}(\lambda) - \mu_t(\lambda)) = \sqrt{m}\sigma_{F,t}(\lambda)I. \tag{6}$$

An adjusted selection intensity can be found from (6), which is equal to $I/\sqrt{3}$. This quantity can be used with (4) to predict convergence time for this problem.

$$t_{conv} = \frac{-\ln 2}{\ln\left[1 - \frac{I}{\sqrt{3m}}\right]}\lambda$$

For tournament selection with a tournament size of two, the selection intensity, I, is $1/\sqrt{\pi}$ and the time to converge the entire string is given in (7). In this equation, k is the string length of each subfunction, not the entire string length. Knowing that $\ln(1-x) \approx -x$ for small x, the convergence time is expressed as

$$t_{conv} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi}}\right]}k = c_t\sqrt{m}\,k \tag{7}$$

where $c_t \approx \ln 2\sqrt{3\pi} = 2.13$. The convergence rate is O($\sqrt{m}k$) for these types of functions.

Using an efficient discretization adds error to the fitness evaluations and causes the GA to take longer to converge. From (3), it is known that $\sigma^2_{F,t}$ becomes $\sigma^2_{F,t}/\sqrt{\sigma^2_{F,t} + \sigma^2_{E,t}}$ in these situations. If binary tournament selection is considered and $r{=}\frac{\sigma^2_{E,t}}{\sigma^2_{F,t}}$ is approximately true during the GA, then (6) becomes

$$m(\mu_{t+1}(\lambda) - \mu_t(\lambda)) = \sqrt{m}\frac{\sigma^2_{F,t}}{\sqrt{\sigma^2_{F,t} + \sigma^2_{E,t}}}(\lambda)I$$

and

$$(\mu_{t+1}(\lambda) - \mu_t(\lambda)) = \sigma_{F,t}\frac{1}{\sqrt{m(1+r)}}(\lambda)I$$

Hence, the adjusted selection intensity is $I/\sqrt{m(1+r)}$. If this is used in the convergence time equation, the convergence time prediction becomes

$$t_{conv} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi(1+r)}}\right]}k = c_t\sqrt{m(1+r)}\cdot k \tag{8}$$

where $c_t \approx \ln 2\sqrt{3\pi} \approx 2.13$ as before. The convergence rate is O($\sqrt{m(1+r)}k$) for these types of functions. When accurate fitness evaluations are used, then (8) is identical to (7).

When error is present in the fitness evaluations, the GA will take longer to converge. The additional time can be determined by considering that the rate that each generation will improve depends on the amount of error in the fitness evaluations.

The ratio of the convergence times can be determined from taking the ratio of (7) to (8). The ratio can thus be written as

$$\frac{t_{conv-accurate}}{t_{conv-error}} = \frac{c_t\sqrt{m}k}{c_t\sqrt{m(1+r)}k} = \frac{1}{\sqrt{1+r}} \tag{9}$$

Since $r \geq 0$, the ratio in (9) is never more than 1. Therefore, a GA with an accurate fitness function is on average expected to converge faster than its error-prone counterpart.

4.4 Population Sizing

Several population-sizing models have been derived for problems with equally salient building blocks, and more recently, population requirements have been considered for problems with exponentially-scaled building blocks. Initially, the population size for exponentially-scaled building blocks was observed empirically. When considering the drift model, it has been observed that the population size varies in the same way that the convergence time varies. Lobo, Goldberg, and Pelikan (2000) considered a single, exponentially-scaled variable that converges as O(l) and also empirically observe that the population size must vary as O(l) as well.

Rothlauf (2001) developed a population-sizing model for problems with exponentially-scaled building blocks by considering the drift model. Drift only affects the GA if $t_{drift} < t_{conv}$, and if drift occurs, the least salient bits will randomly converge to either 0 or 1. If the bit string of length *l* is composed of *m* concatenated sub-strings of length k, each of which is exponentially-scaled, the lower bound of the population size given by the drift model when using tournament selection of size 2 can be determined if $t_{drift} = t_{conv}$:

$$N_{drift} = \frac{5\pi}{14}\sqrt{\frac{\pi}{m}}l \tag{10}$$

where $m \cdot k = l$. This model implies that the population size varies as O($k\sqrt{m}$). If the population is inadequately sized (i.e. $N < N_{drift}$), then drift is probable. Previously, it was shown that a GA without evaluation error will converge as O($k\sqrt{m}$). This implies that when evaluation error is present in the GA, the population size must vary as O($k\sqrt{m(1+r)}$), where $r = \frac{\sigma^2_{E,t}}{\sigma^2_{F,t}}$, because the GA converges at the slower rate of O($k\sqrt{m(1+r)}$). These observations imply that the convergence time and population size varies in the same way for exponentially-scaled problems.

This model does not take into account building-block mixing, and it gives inadequately-sized populations. The population size equations can be rewritten as

$$N_n = c_N\sqrt{m}k,$$

$$N_e = c_N\sqrt{m(1+r)}k,$$

and c_N can be found empirically once *r* is known.

5 Discretization Scheduling in one Dimension

In this section, the components from Section 4 are brought together for the time budgeting of a genetic algorithm in one-dimension. These components are used to derive an efficient schedule of discretization that decreases the total computation time for a GA.

This section begins with the introduction of an idealized one-dimensional problem. Although the derivations and methods are developed after the example problem is presented, the schedule of discretization and the predicted speedup is valid for all such problems in one-dimension. Time budgeting has three parts: discretization scheduling, population sizing, and the estimated number of generations until convergence. The last two parts were discussed in Section 4, this section is devoted to discretization scheduling and estimating the computational time requirements for each function evaluation as well as the entire GA. Following this, the idealized experiments show the speedup from using an efficient discretization.

The example problem considered is the integral of the derivative of a sum of m independent and identical subfunctions. If the integration is done precisely enough, the optimal value of the GA is the same as the optimal value of the sum of the subfunctions themselves. The experiments in this section have the following form where f is the fitness function:

$$f = \sum_{i=1}^{3} \int_0^{x_i} \frac{d}{dx_i} g_i(x_i) dx_i$$

$$g(x) = e^{Ax} \cos Bx$$

In this equation, A and B are 0.05 and 2, respectively. Ten bits are used to represent the string for each x in $g(x)$. The maximum of each subfunction occurs when x is 9.44 when x varies from 0 to 10.23, and there are a total of four local optima.

For any number of grid points n, it is expected that phase shift δ is smaller than the size of the discretization h for any arbitrary n higher than some usable level. The size of the discretization $h = l/n$ when the grid points are equally spaced and l is the total length considered ($l = 10.23$ for this example). Figure 3 shows how δ varies with n. The phase shift decreases rapidly and it is smaller than h when n is 5 or larger. This indicates that even if a function has several peaks, a small number of grid points is needed in order for the GA to lead the approximate solution to within h of the actual optimum.

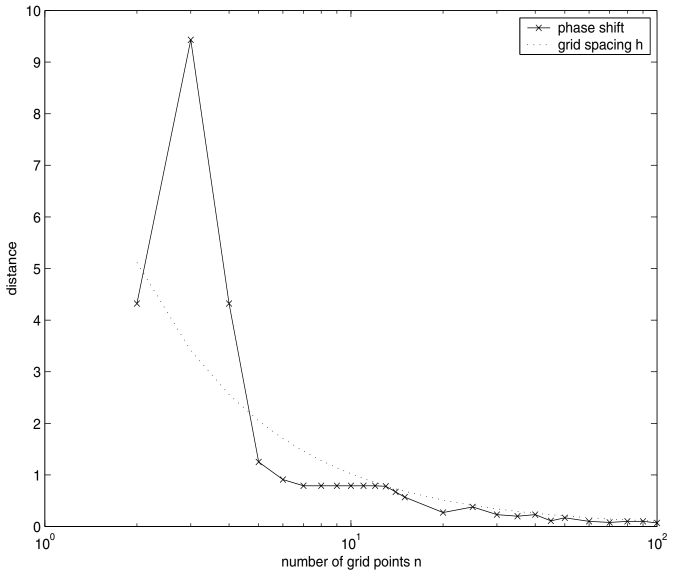

Figure 3: The phase shift for the idealized one-dimensional experiments.

In order to insure that the first building blocks do not start to converge to the wrong values, n must be at least 5. In the first generations, 8 grid points are used, the next highest power of 2 larger than 5. This corresponds to *usable* error as discussed in Section 4, which states that the phase shift must be less than the grid size in order for the GA to make correct decisions.

5.1 Time Budgeting for a Naive Discretization

A naive discretization assumes that the discretization is constant throughout the duration of the GA. This can be contrasted with an efficient discretization, to be introduced in Section 5.2, in which the discretization varies within the GA.

The computation time for a GA with a sampling fitness function (Fitzpatrick and Grefenstette, 1988) can be modeled as

$$T = (\alpha + \beta n) t_{conv} N, \tag{11}$$

where α is the overhead per individual per generation, β is the time to calculate one sample, n is the number of samples, and N is the population size. The term $\alpha + \beta n$ is the cost of each function evaluation. This equation can be used to model the computation time of a fitness function considering discretization in one-dimension, where n is the number of grid points used as long as the previous assumption holds, that the computational time of the integration depends only on n.

After substituting the convergence time from (7), (11) can be rewritten as

$$T = (\alpha + \beta n) c_t \sqrt{m} k N_n \tag{12}$$

when t_{conv} is replaced by its equivalent, $c_t \sqrt{m} k$ from (7).

5.2 Time Budgeting for an Efficient Discretization

When the number of grid points n is fixed within the GA, n is too precise at the beginning of the run, which leads to wasted computation cycles early on. A more efficient discretization would be to use fewer grid points in the first generations and increase the number of grid points throughout the GA because domino convergence indicates that in the first few generations, the GA will be considering only the most salient bit. In theory, only two grid points are needed, because in the first few generations, if the GA is able to correctly choose the value of this building block, then only two grid points are needed by the fitness function in every dimension. In practice, the smallest number of grid points that are needed at the beginning of the GA may be larger than 2 in order to get a desired level of accuracy in the fitness function.

GA efficiency can be improved if only as many grid points are used as necessitated by the accuracy of the numerical scheme used by the GA at any given point in time. Since it is known which building blocks are converging at what time, how the grid should be spaced can be determined in order to efficiently converge to that particular building block. That is, if two grid points are initially used by the GA, the number of grid points must be doubled in order to converge to the second bit. Thus, increasing the number of grid points exponentially will match the salience of the converging bits when the bits are converted to a floating point number.

A certain number of grid points are needed to differentiate between two competing individuals such that the GA can correctly determine the value of the least salient building block in the string. For example, if a string of length l has exponentially-scaled building blocks, 2^l grid points are needed to differentiate for the numerical integration. If fewer grid points are used, then the integration cannot discriminate between all possible string values.

Assuming that the usable number of grid points starts at 2 and is doubled every $c_k = c_t \sqrt{m(1+r)}$ generations, the time for any bit in each subfunction to converge (see (8)), then the efficient computation time in a one-dimension equation can be rewritten in an efficient form of

$$T_e = \sum_{i=1}^{k} (\alpha + \beta \cdot 2^i) c_k N_e$$

Since the efficient population size can be written as $N_e = N_n \sqrt{1+r}$, the previous equation can be rewritten as

$$T_e = \sum_{i=1}^{k} (\alpha + \beta \cdot 2^i) c_t \sqrt{m} N_n (1+r) \tag{13}$$

In (13), it is assumed that there is a direct relationship between the number of grid points and the precision to which the function values can be differentiated. Because of this relationship, for the GA to discriminate between two individuals up to a value of $1/2^j$ at any time with *j* grid points, then 2^k grid points are needed in the final stages of the GA. If there is a less direct relationship between the number of grid points and the level of discrimination in the string, their efficient computation time equations are easily derived.

5.3 Speedup

Speedup is a measurement of computational savings when using an efficient discretization compared to a naive discretization. Using an efficient discretization results in fitness evaluations that are inexpensive in early generations and expensive in later generations. A naive discretization has expensive fitness evaluations in every generation. The speedup can be measured as the naive time divided by the efficient time.

$$S = \frac{T_n}{T_e} \tag{14}$$

Although the efficient GA takes more generations to converge than its naive counterpart, the overall savings from using an efficient discretization throughout the GA is significant. In (15), it is shown how speedup varies with different values of α/β. Computational experiments empirically show the computational savings. When using the naive and efficient computation times as in (12) and (13), speedup can be generalized as

$$S = \frac{\alpha k + \beta n k}{\alpha k + \beta \sum_{i=1}^{k} 2^i} \cdot \frac{1}{1+r} \tag{15}$$

In (15), the population size and the number of subfunctions, *m*, are not present and it does not affect the computational savings. Since $n = 2^k$ and $\sum_{i=1}^{k} 2^i = 2^k - 2$, then (15) is simplified to

$$S = \frac{\frac{\alpha}{\beta} k + k 2^k}{\frac{\alpha}{\beta} k + (2^k - 2)} \cdot \frac{1}{1+r}. \tag{16}$$

If there is no apparent error variance (i.e., $r = 0$), then the speedup can be predicted for various values of α/β and variable lengths, *k*. The predictions are shown in Figure 4. This figure shows that the speedup when *k* is low depends on α/β. When *k* is larger, α/β has a negligible effect on the speedup, and the speedup increases linearly for larger values of *k*. Asymptotically, the speedup is equal to the sub-length *k*.

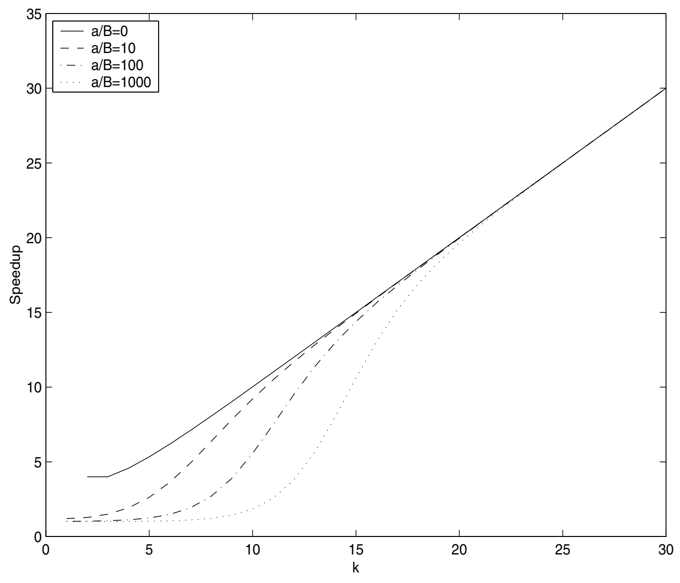

Figure 4: Speedup for different values of α/β with $r = 0$.

5.4 Idealized One-Dimensional Experiments

For the computational experiments, the rectangle rule—the left-endpoint rule—is used, and it is O(h) accurate (Dahlquist, 1974). As the problem was introduced at the beginning of this section, the convergence time and various GA parameters are introduced. Finally, the results are presented.

The apparent error variance can be estimated from analyzing the change in fitness when switching grids. The value of *r* does not exist in the first few generations before the grid spacing is changed, and near the end of the GA, *r* becomes very large as the fitness variance approaches zero. Figure 5 shows the ratio of the error variance to the fitness variance during the GA, and it verifies that *r* is approximately constant and is equal to 0.90. The value of *r* is not entirely constant, and the jumps in the values of *r* are due to the abrupt changes in the discretization.

5.4.1 GA parameters and assumptions

The specific assumptions regarding the operators of simple genetic algorithms are briefly stated. Aside from the specific convergence time and population size requirements, these assumptions are also used for the experiments in Sections 6 and 7.

Each individual is composed of a bit string of length *l*, which is a vector of decision variables. If it is assumed that the string is composed of m variables of length *k*, then $l = mk$. Each string represents one of the 2^l possible solutions and has an associated scalar objective function value that represents the fitness of the individual.

Pairwise tournament selection is used for all experiments, and all individuals participate in selection. One-point crossover is used on a pair of individuals—the parents—to create two more individuals—their children. Every individual experiences crossover; none are copied over into the new generation. The probability of mutation is $1/N$ (De Jong, 1975). In all experiments, each population in the current generation is completely replaced by another population of the same size in the next generation. The

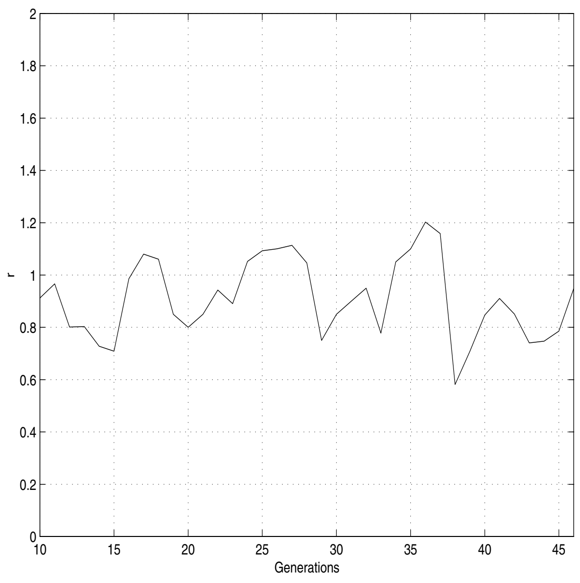

Figure 5: The value of *r* for the idealized one-dimensional experiments.

process is repeated until the stopping criteria is met.

Gene convergence is used as the termination criteria for halting the GA in the experiments. Gene convergence halts the GA after the individuals have converged to a certain proportion of their correct values. For the experiments, a 97-percent convergence rate is required for termination.

The convergence time of the example function can be predicted by the convergence time equations in Section 4.3. The naive and efficient convergence time equations are rewritten.

$$t_{conv-naive} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi}}\right]} k = c_t \sqrt{m} k$$

$$t_{conv-efficient} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi(1+r)}}\right]} k = c_t \sqrt{m(1+r)} \cdot k$$

It is known that m is 3 and k is 10, and $r = 0.90$. The naive and efficient convergence times are 33.3 and 47.3 generations, respectively. For the efficient experiments, the grid points will be doubled every $c_k = 4.73$ generations, where $c_k = c_t\sqrt{m(1+r)}$.

The population size can be inferred from the work of Rothlauf (2001), who predicts that the population size for problems considering exponentially-scaled building blocks varies as O($k\sqrt{m}$), where m is the number of building blocks and k is the length of each building block. The naive and efficient population sizes can be expressed as

$$N_n = c_N \sqrt{m} k$$

$$N_e = c_N \sqrt{m(1+r)} k$$

Experiments when using a naive discretization indicate that $N_n = 150$ individuals and, therefore, $c_N = 8.66$. Since $r = 0.90$, the efficient population size is 206 individuals.

5.4.2 Time budgeting and speedup

Once all the parameters have been set, the computational requirements can be predicted for both the naive and efficient implementations.

First, α and β must be determined for the time budgeting equations, (12) (13). For the example problem, computational experiments indicate that the values for α and β are 3.3×10^{-4} and 2.6×10^{-6}, respectively, when the number of floating point operations are used to measure computation time.

As mentioned previously, 8 grid points are used in the initial population to ensure that a good solution is found by the GA. In other words, the first $3 \cdot c_k = 14.2$ generations are run with 8 grid points, and after that, the number of grid points are doubled every $c_k = 4.73$ generations. This has a negligible effect on computation time because the ratio to βn to α is small when n is 8.

Substituting in the naive parameters, we can predict the total computation time to be 14.9×10^6 floating point operations (14.9 Mflops) and 5000 total function evaluations. The efficient implementation is predicted to use 8.4 Mflops and 9740 function evaluations. This yields a predicted speedup of 1.78. The results of the experiments can be seen in the next section.

5.4.3 Results

A total of 50 trials were run with both the naive and efficient discretizations. As mentioned earlier, a naive time of 14.9 Mflops and an efficient running time of 8.4 Mflops are expected, or a speedup of 1.78. Tables 1 and 2 summarize the expected and actual generations, time, and number of function evaluations until convergence.

Table 1: Predicted and actual computation time values for one-dimensional experiments using a naive discretization.

	Generations	**Time (Mflops)**	**Function Evaluations**
Predicted:	33.3	14.9	5000
Actual:	33.7	15.1	5060

Table 2: Predicted and actual computation time values for one-dimensional experiments using an efficient discretization.

	Generations	**Time (Mflops)**	**Function evaluations**
Predicted:	47.3	8.4	9740
Actual:	47.7	8.7	9830

In both cases, the actual number of function evaluations and generations are close to their predictions, and in the case of naive discretization, the computation time prediction follows very closely to the theory. When an efficient discretization was used, the actual computation time was longer than the predicted because there is a large penalty for every evaluation over that which was predicted because the fine-grained discretization is used for these extra generations. Despite the large penalty for taking longer to converge, and the fact that the efficient runs required almost twice as many total evaluations, a speedup of 1.74 was observed, close to the 1.78 that was predicted.

An accurate measure of performance is to see the building blocks converge with

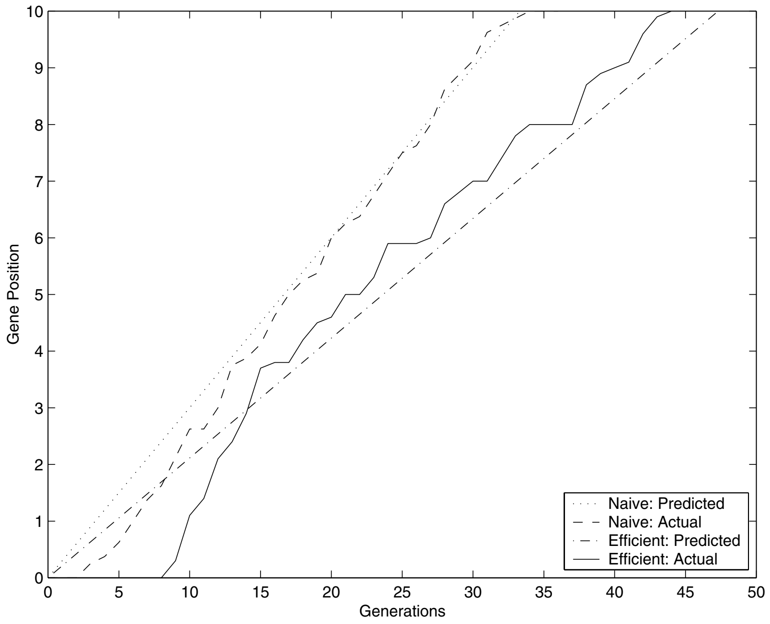

Figure 6: Predicted and actual gene position for idealized one-dimensional experiments.

time. Because of domino convergence, the gene position, or number of genes that have converged in each subfunction, is expected to vary linearly with time. Figure 6 shows the average converged gene position for the trials. In both the naive and efficient cases, the experimental gene position follows very closely to the predicted value. It should be noted that although the efficient discretization requires almost twice as many function evaluations to converge, it converges in nearly half the time as compared to the naive discretization.

6 Discretization Scheduling in Multiple Dimensions

In Section 5, efficient GAs were developed using discretization scheduling in one-dimension. In this section, the results are generalized in multiple dimensions, and idealized experiments are run in two-dimensions.

As in Section 5, this section begins with the introduction of an idealized problem. The schedule of discretization and the predicted speedup is valid for all such problems in multiple dimensions. Following the derivation of time budgeting and speedup equations, the idealized experiments are run to show the speedup from using an efficient discretization.

The fitness function considered is an integral in two dimensions. The experiments have the following form where f is the fitness function:

$$f = \sum_{i=1}^{3} \int_0^{x_i} \int_0^{y_i} \frac{\partial^2 g_i(x_i, y_i)}{\partial x_i \partial y_i} \partial y_i \partial x_i \tag{17}$$

$$g(x, y) = (e^{Ax} \cos(Bx))(e^{Ay} \cos(By)) \tag{18}$$

The fitness function is the sum of three identical subfunctions, where A and B are 0.1 and 1.4, respectively. Six bits are used to represent the string for each x and y variables when the variables are mapped from 0 to 10.08. Thus, the total chromosome length is 36 bits. The maximum occurs at 6.88 for each x and each y in each subfunction.

The integration ideally should start with two grid points in each dimension. As in the one-dimensional case, two grid points are inadequate for the integration because the function is multimodal. An analysis shows that at least 8 grid points are needed in each dimension for the GA to choose the correct first bit. When fewer grid points are used, the GA starts to converge to a local optimum and correct bits are lost.

6.1 Time Budgeting for a Naive Discretization

When the discretization occurs in d dimensions, the naive time budgeting in (11) can be generalized as

$$T = (\alpha + \beta n_1 n_2 ... n_d) t_{conv} N_n.$$

When the number of grid points in any dimensions is identical and $t_{conv} = c_t\sqrt{m}k$, the above equation can be simplified to

$$T_n = (\alpha + \beta n^d) c_t k \sqrt{m} \cdot N_n. \tag{19}$$

6.2 Time Budgeting for an Efficient Discretization

The efficient time budgeting equation in one dimension can be extended to predict the efficient time budgeting in two or more dimensions. It is assumed that the discretization is scheduled in multiple dimensions in the same way that it is scheduled in one-dimension, by starting with two grid points in each dimension and doubling the number of grid points every c_k generations, the time to converge to the ith bit of each subfunction. The assumption that only two discretizations are needed in each dimension may be insufficient for the GA to converge to the correct optima, so a larger number of grid points may be needed to start with. It is assumed that at any given time, the number of grid points in each dimension is the same, but an efficient time budget can be easily derived when the number of grid points varies in every dimension.

For an arbitrary number of dimensions, d, the efficient time is

$$T_e = \sum_{i=1}^{k} (\alpha + \beta \cdot 2^{d \cdot i}) c_k N_e.$$

The above equation is written, after substituting $N_e = N_n\sqrt{1+r}$ and $c_k = c_t\sqrt{m(1+r)}$, as

$$T_e = \sum_{i=1}^{k} (\alpha + \beta \cdot 2^{d \cdot i}) c_t \sqrt{m} N_n (1+r). \tag{20}$$

6.3 Speedup

The speedup can be analyzed as was done in Section 5. The speedup can be written in terms of the naive and efficient computation times:

$$S = \frac{T_n}{T_e} = \frac{(\alpha + \beta n^d) c_t \sqrt{m} k \cdot N_n}{\sum_{i=1}^{k} (\alpha + \beta \cdot 2^{d \cdot i}) c_t \sqrt{m} \cdot N_n (1+r)}$$

This equation can be simplified and rewritten since $\sum_{i=1}^{k} 2^i = 2^k - 2$ and $n = 2^k$.

$$S = \frac{\frac{\alpha}{\beta}k + k2^{k\cdot d}}{\frac{\alpha}{\beta}k + (2^k - 2)^d} \cdot \frac{1}{1+r}. \tag{21}$$

As in the two-dimensional case, the speedup can be predicted for different values of α/β and k. When predicted to the same predicted speedup than in Section 5, it can be seen that larger speedups are attained for identical values of *k* because of the additional expense for each evaluation in two-dimensions.

If there is no apparent variance error (i.e., $r = 0$) and there are the same number of grid points in two-dimensions, then the speedup can be predicted for various values of α/β and variable lengths, *k*. The predictions are shown in Figure 7. This figure shows that the speedup when *k* is low depends on α/β, but for moderate or large values of *k*, the speedup increases linearly. The speedup appears to be less dependent on α/β than in the one-dimensional case. Asymptotically, the speedup is equal to the sub-length *k*.

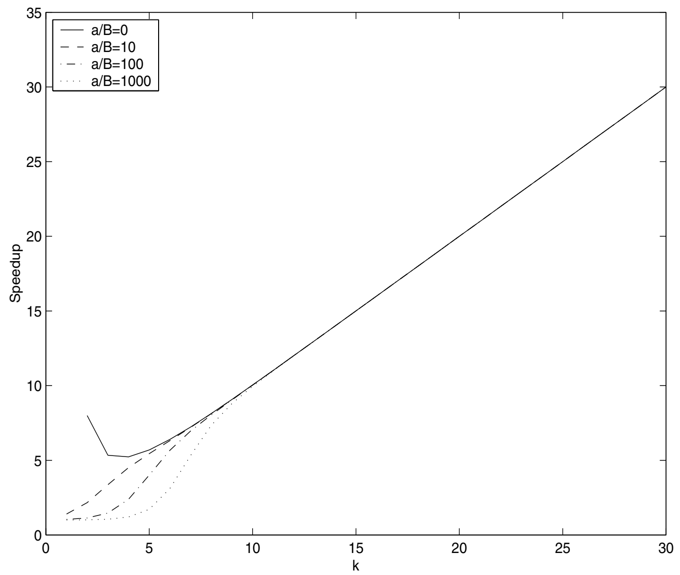

Figure 7: Predicted speedup for a two-dimensional problem.

6.4 Idealized Two-Dimensional Experiments

In this section, the setup and results of the two-dimensional idealized experiments are given. As in the one-dimensional case, the left-endpoint rule is used for the numerical integration method.

As in the one-dimensional case, switching the grid results in apparent variance error. A minimum of 8 grid points is needed for the GA to find the correct solution. When switching from 8 to 16 grid points, apparent variance error is added to the fitness evaluations. Figure 8 shows how *r* varies within the GA. The value of the variance is approximately 0.80 and it is roughly constant during the GA. This value of *r* is used to predict the convergence time and the population size.

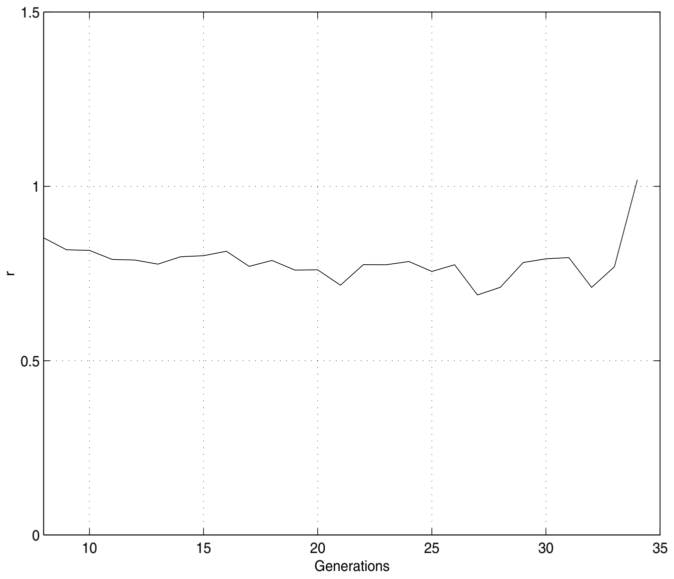

Figure 8: The value of *r* for the idealized two-dimensional experiments.

6.4.1 Convergence time analysis

The convergence time equation must be modified to take into account the three subfunctions and two variables in each subfunction. Since there are three subfunctions, then the predicted change in population mean is the same as in the one-dimensional case. If the subfunction can be written as $g(x, y) = h(x) \cdot h(y)$, then the equation

$$\mu_{t+1} - \mu = \sigma_{F,t} I$$

is valid as written for $h(x)$ and $h(y)$, where $h(x)$ and $h(y)$ are identical when $x = y$. In the same way, $m' = m^2$, which adjusts for $g(x, y)$. Since m is 3, m' is 9:

$$m'(\mu_{t+1}(\lambda) - \mu_t(\lambda)) = \sqrt{m'}\sigma_t(\lambda) I \tag{22}$$

In other words, the adjusted selection intensity can be written as $I' = I/\sqrt{m'}$. If binary tournament selection is considered, then the convergence times are the same as (7) and (8) with $m' = 9$ used instead of m:

$$t_{conv-naive} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m'\pi}}\right]} k = c_t \sqrt{m'} k$$

$$t_{conv-efficient} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m'\pi(1+r)}}\right]} k = c_t \sqrt{m'(1+r)} \cdot k$$

Using these values, the naive convergence time is 36.2 generations. Knowing that *r* is 0.80, the efficient convergence time is expected to be 49.2 generations.

6.4.2 GA parameters

As in the one-dimensional experiments, the population size was empirically determined by the following equations:

$$N_n = c_N k \sqrt{m'}$$

$$N_e = c_N k \sqrt{m'(1+r)}$$

This results in a c_N of 8.89 and a population size of 160 individuals. The convergence time for the efficient experiments is 215 individuals when $r = 0.80$. The drift time for the naive and efficient experiments are 224 and 301 generations, much longer than the predicted convergence times of 36.2 and 49.2 generations. Hence, drift is not a concern for these experiments.

For the experiments, the probabilities of selection and crossover are 1.0, and the probability of mutation is $1/N$. The rest of the assumptions are the same as in the one-dimensional experiments (see Section 5.4.1).

6.4.3 Time budgeting and speedup

The expected computation time can be found when α and β are known. Here, the two dimensional case is considered. Computational experiments indicate that α and β are 2.51×10^{-5} and 1.37×10^{-7} respectively when Mflops are used to estimate the computation time. To be able to discriminate between the least salient bit in each of the subfunctions, 64 grid points are needed in each dimension. Then, (19) and (20) predict the time needed for the naive and efficient implementations.

The naive computation time is 3.40 Mflops and the number of function evaluations necessary is predicted to be 5790. For the efficient case, 1.59 Mflops are needed when the multiple-dimension time budgeting equation is considered and 10,580 function evaluations are necessary. This yields a predicted speedup of 2.14. As mentioned previously, the GA is started with 8 grid points in each dimension. In other words, the first $3 \cdot 8.20 = 24.6$ generations are run with 8 grid points in both the x and y dimensions.

6.4.4 Results

For the experiments, 50 trials were run using both the naive and efficient discretization. The expected speedup from using an efficient discretization is 2.14. Tables 3 and 4 show the generations, time, and number of fitness evaluations for the naive and efficient implementations.

Table 3: Predicted and actual computation time values for two-dimensional experiments using a naive discretization.

	Generations	**Time (Mflops)**	**Function evaluations**
Predicted:	36.2	3.40	5790
Actual:	37.1	3.48	5940

As in the one-dimensional case, the efficient GA takes slightly longer to converge—50.0 generations compared to the 49.2 predicted generations. The actual speedup of 2.06 is very close to the predicted speedup of 2.14. That is, despite the efficient GA making almost twice as many function evaluations as the naive GA, on average, it finished in less than half the time. Figure 9 shows the converged gene position for each subfunction for the naive and efficient implementation. Like in the one-dimensional

Table 4: Predicted and actual computation time values for two-dimensional experiments using an efficient discretization.

	Generations	Time (Mflops)	Function evaluations
Predicted:	49.2	1.59	10,580
Actual:	50.0	1.69	10,750

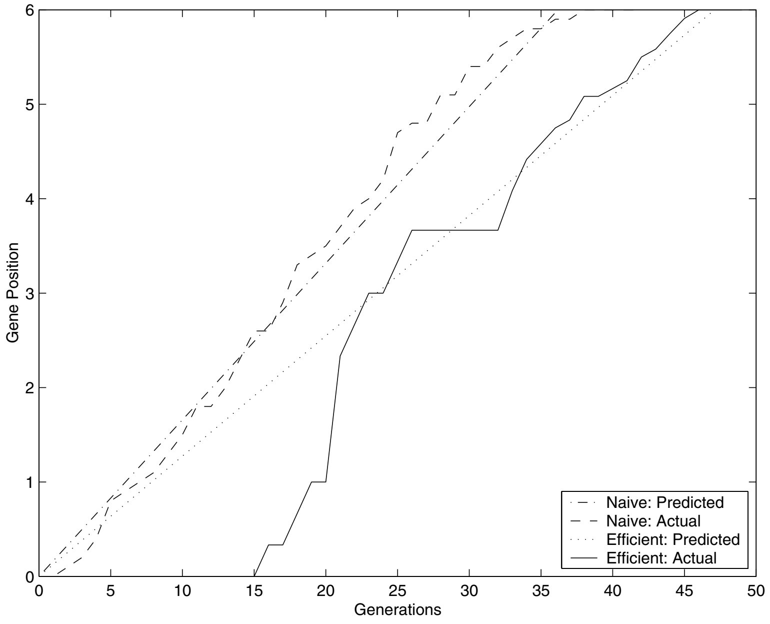

Figure 9: Predicted and actual gene position for idealized two-dimensional experiments.

experiments, the experimental gene position follows closely to the predicted gene position.

7 Inverse Groundwater Application

In this section, inverse problems in groundwater are introduced and a groundwater application is modeled and solved using a genetic algorithm. The improvement in performance is shown when using an efficient discretization.

Inverse groundwater problems are useful when the exact parameters of a problem—in this case, the transmissivity values—are unknown, and the observations made in the field are used to estimate these parameters. These problems are modeled by setting up a finite differencing grid in time of a one- or two-dimensional region, and there is an initial guess for the parameters which are adjusted until the error between the observed and calculated head values is minimized. Groundwater problems can be difficult to accurately model either because the equations may not be representative of the original problem or because parameters of the original problem, such as geometry, boundary conditions, sink and source terms, are difficult to measure and are a source of error for the model (Sun, 1994).

Well-posed and unique solutions to an inverse groundwater problem often do not exist, and there are several issues regarding the modeling of groundwater problems that can complicate finding a unique physically meaningful solution. Although solutions to inverse groundwater exist, realistic solutions may not exist because of error in the observations. Additionally, it has long been understood that inverse groundwater modeling is often unstable. Although solutions always exist, the uniqueness is often not guaranteed as several combinations of parameters may lead to similar observations, or there may be several optima that are virtually equivalent in their objective values. There are several ways in which uniqueness can be guaranteed, although the information needed is typically expensive to obtain (Sun, 1994).

Genetic algorithms have been used to solve groundwater problems (Harrouni et al., 1996; Gwo et al., 2000; Karpouzos et al., 2001). These authors find that GAs have an advantage over the indirect method in that GAs do not require gradient information, and that GAs can be run in parallel to decrease required computation time. In addition, El Harrouni et al. (1996) empirically observe that GAs do not get trapped in local optima as often as their traditional counterparts.

7.1 Two Dimensional Flow in a Confined Aquifer

In this section, the particular methodology used for the inverse groundwater problem is first introduced, including the governing equations and quadrature. For the experiments, a two-dimensional flow in a confined aquifer is considered. The flow is assumed to move from left to right only, and there is no other flow across the top and bottom boundaries. A block centered grid is used where the left and right edges are equal to the boundary conditions, and the center of each block is equal to one of the head calculations. The grid has the same number of grid points in both dimensions at all times, with the parameters calculated for the center of each block.

The governing equation for two-dimensional flow in a confined aquifer is

$$\nabla \cdot (T \nabla h) + q = 0 \tag{23}$$

where T is the transmissivity, q is the sink/source term, and h is the head. Then, (23) can be written in a 5-point form from Crank Nicholson time differencing by using a point and the four surrounding points. The right and left endpoints are the boundary conditions of the region.

$$\begin{aligned} AX_{i+1/2,j}h_{i+1,j} + AX_{i-1/2,j}h_{i-1,j} + AY_{i,j+1/2}h_{i,j+1} + AY_{i,j-1/2}h_{i,j-1} \\ -(AX_{i+1/2,j} + AX_{i-1/2,j} + AY_{i,j+1/2} + AY_{i,j-1/2})h_{i,j} = 0 \end{aligned} \tag{24}$$

where *AX* and *AY* are given by

$$AX_{i+1/2,j} = \frac{(T_x)_{i+1/2,j}\Delta y_j}{x_{i+1} - x_i} \tag{25}$$

$$AY_{i,j+1/2} = \frac{(T_y)_{i,j+1/2}\Delta x_i}{y_{j+1} - y_j}. \tag{26}$$

In (24), (25), and (26), x_i and y_i are the locations of grid points in the two dimensions, where Δx and Δy are the grid spacing in the x and y dimensions, respectively. Likewise, $(T_x)_{i,j}$ and $(T_y)_{i,j}$ are the transmissivity values at the *i*th grid point in the x-dimension and the *j*th grid point in the y-dimension.

The fitness function is the sum-squared difference between the observed and estimated head observations. In many cases, the position of the head measurements do not line up exactly with the grid points. In this case, an approximated fitness is made from linearly interpolating the head measurements from the two or four surrounding head measurements, depending on where the head observation is with respect to the blocks in the grid. The fitness function f can thus be written as a sum over all of the head observations

$$f = \sum_{i \in Obs.} w_i (H_i - \hat{H}_i)^2 \tag{27}$$

where H_i is the ith head observation, $\hat{H}_i$ is the corresponding ith head measurement as calculated with finite differences, and w_i is the weighting factor.

7.2 Time Budgeting for a Naive Discretization

The naive computation time generalizes the multidimensional computation time presented in Section 6.1 in two ways. First, these problems consider a general, polynomial-time relationship between the grid spacing and computation time, where grid spacing and time are related by a factor p. In addition, the previous equations assume that the grid spacing and the size of the problem are related by $n = 2^k$. For problems using finite differences or finite elements, there may not be a direct relationship between these two quantities. In this case, $k' = \log_2 n$ is considered, where n is the size of the grid as chosen by the GA practitioner. This generalizes the derivations in Sections 5 and 6 to all types of fitness functions, and it may be more useful for the finite-difference and the finite-element methods, which may require the solving of a large system of linear equations with Gaussian elimination or LU-decomposition, for example.

The naive computation time equation can be generalized for this condition as

$$T = (\alpha + \beta n_1^p n_2^p ... n_d^p) t_{conv} N.$$

When the number of grid points is the same in every dimension, this equation can be simplified to

$$T_n = (\alpha + \beta n^{d \cdot p}) c_t \sqrt{m} k \cdot N_n \tag{28}$$

when $p = 1$, (28) reduces to (19), the multi-dimensional naive time complexity equation.

7.3 Time Budgeting for an Efficient Discretization

The efficient computation time can be generalized for the situation when there is a polynomial-time relationship between time and grid spacing, and when the number of grid points n may not be equal to 2^k. In this case, the number of grid points is doubled every $c_{k'} = c_k \frac{k}{k'}$ generations, where $c_{k'} = c_t \sqrt{m(1+r)} \frac{k}{k'}$. Then, the efficient computation time can be written as

$$T_e = \sum_{i=1}^{k'} (\alpha + \beta \cdot 2^{p \cdot d \cdot i}) c_{k'} N_e.$$

This equation can be rewritten as

$$T_e = \sum_{i=1}^{k'} (\alpha + \beta \cdot 2^{p \cdot d \cdot i}) c_t \frac{k}{k'} N_n (1 + r). \tag{29}$$

As p increases ($p \geq 1$), and the computational requirements become increasingly more expensive for an increasing number of grid points.

7.4 Speedup

The speedup is expressed in terms of the naive and efficient computation times. When using the most general naive and efficient computation times in (28) and (29) above, assuming that the number of grid points in any dimension is the same, speedup can be expressed as

$$S = \frac{(\alpha + \beta n^{d \cdot p}) c_t \sqrt{m} k N_n}{\sum_{i=1}^{k'} (\alpha + \beta \cdot 2^{p \cdot d \cdot i}) c_t \frac{k}{k'} N_e}.$$

This can be simplified to

$$S = \frac{\frac{\alpha}{\beta} k + k 2^{k' \cdot d \cdot p}}{\frac{\alpha}{\beta} k' + (2^{k'} - 2)^{d \cdot p}} \cdot \frac{1}{1+r}. \tag{30}$$

In (30), the population size and the number of subfunctions, m, are not present and do not affect the computational savings.

7.5 Groundwater Experiments

In this section, the specific problem and its parameters are defined, which include boundary conditions, zoning, and head observations. After this, the GA parameters are derived from the problem parameters, the computational speedup is predicted, and the results are shown.

7.5.1 Specific parameters

Two-dimensional flow in a contained aquifer of a rectangular shape is considered. These parameters are illustrated by Figure 10 and are explained below. The flow moves from left to right only, and the size of the domain is 10 m in the x-dimension and 8 m in the y-dimension. The boundary conditions for the head are 1.0 m at the left boundary and 0.5 m at the right boundary. The equally spaced head observations are placed throughout the field and are shown by the circles in Figure 10. The zones are divided into four equally spaced rectangular regions, and each region has its own transmissivity value. The simple geometry and boundary conditions imply that this problem is not ill-posed.

The head observations are generated by the forward method using the exact transmissivity values. Four rows of 16 equally spaced head observations made (a total of 60 observations) at $y = 1.25\ m$, $y = 6.75\ m$, $x = 1.5635\ m$ and $x = 8.4375\ m$ are used to estimate the sum-squared error between the observations and the predictive model. The exact head measurements reduce the degree of non-uniqueness because head observations are typically inaccurate, and this variability may cause multiple solutions to exist. Since a relatively large number of observations are used to estimate the fitness, non-uniqueness is essentially not a factor.

According the zoning in Figure 10, the GA solves for four transmissivity values. Each transmissivity is composed of 8 bits ($k = 8$), which is mapped from 4.0×10^{-8} to $1.024 \times 10^{-5}\ m^2/s$. The optimal transmissivity values are $T_1 = 4.0 \times 10^{-8}$, $T_2 = 9.92 \times 10^{-6}$, $T_3 = 4.0 \times 10^{-8}$ and $T_4 = 1.024 \times 10^{-5}\ m^2/s$.

Since there are 16 equally spaced head observations in the x- and y-dimensions, 16 grid points are used for the naive discretization. In other words, k' for the time budgeting equations is 4 since $k' = \log_2 16$. Because of how the zones are estimated, two grid points can be used in each dimension at the beginning of the efficient implementation, one point for each of the zones.

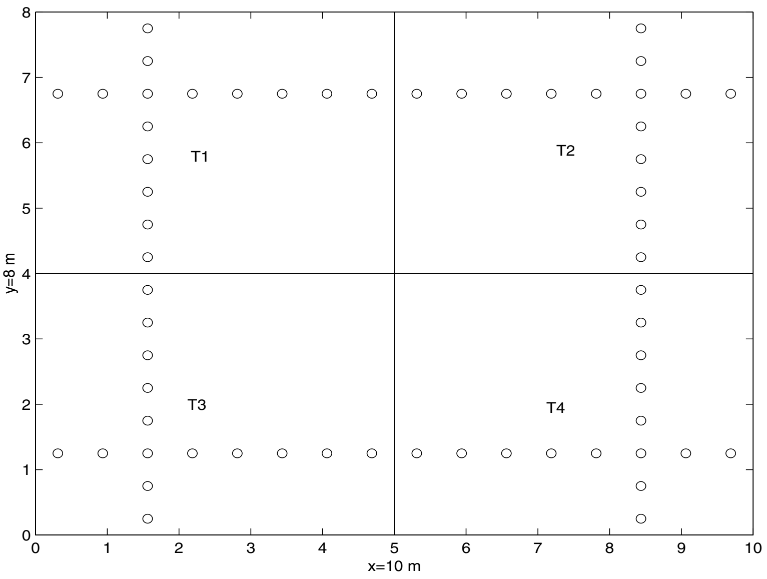

Figure 10: The parameters and zones used in the groundwater problem.

In order for each decision variable, T_i, to converge at the same rate, the observations must make the same contribution to the fitness and have the same variance, or the transmissivity values will start to converge at different times during the GA. As mentioned in Section 6.2, the contribution of each observation can be weighed, and four weights are needed, one corresponding to each zone. The new fitness can be written as:

$$f = \sum_{j=1}^{4} \left(w_j \sum_{i \in Zone\ j} (H_i - \hat{H}_i)^2 \right) \tag{31}$$

The weights are determined for generation $t+1$ by considering the contribution of each zone to the fitness at generation t. The weights are changed with respect to w_1, which is always 1.0. When the fitness values are weighed, both the fitness contributions and variances are approximately equal.

7.5.2 Convergence time analysis

Once the weights have been added as above, the convergence time can be predicted as if each subfunction was identical. As in the idealized one- and two-dimensional experiments, the adjusted selection intensity must be predicted for this problem. Because there are 4 subfunctions, *m* is 4, and the standard deviation is adjusted by a factor of $\sqrt{4} = 2$:

$$4(\mu_{t+1}(\lambda) - \mu_t(\lambda)) = 2\sigma_{E,t}(\lambda)I.$$

The adjusted selection intensity is therefore $I/2$, and this value can be applied directly to the time convergence equations:

$$t_{conv-naive} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi}}\right]} k = c_t\sqrt{m}k$$

$$t_{conv-efficient} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi(1+r)}}\right]} k = c_t\sqrt{m(1+r)} \cdot k$$

From these equations, it is found that in the naive case, the number of generations until convergence is 31.2.

The apparent variance error can be estimated when switching the grids. The ratio is approximately 0.52 during the GA. Thus, the convergence time for the efficient case is 39.2 generations.

7.5.3 GA parameters

The population size is determined empirically for this problem. When the population size is determined for the naive case, the population for the efficient case can be determined with the value of r:

$$N_n = c_N k\sqrt{m}$$

$$N_e = c_N k\sqrt{m(1+r)}$$

The population sizes for the naive and efficient implementations are 220 and 270, respectively. The drift time can be checked to see if such sizes ensure that the drift will not occur. Using the equation $t_{drift} = 1.4N$, drift is predicted to occur at 308 and 378 generations, well after convergence.

For the experiments, the probabilities of selection and crossover are 1.0. The probability of mutation is 1/N. The rest of the GA parameters are summarized in Section 5.4.1.

7.5.4 Time budgeting and speedup

The naive and efficient computation times can be predicted by (28) and (29) once α, β and p are known. Computational experiments indicate that α is 0.00271, β is 2.7×10^{-5} and p is 1.56. Using these parameters, the naive computation time is predicted to be 704 Mflops. As mentioned previously, 16 discretizations are used in each dimension for the naive case. In other words, k' is 4. For the efficient implementation, the number of grid points in each dimension is initially 2, and the number of grid points are doubled every $c_{k'} = c_t\sqrt{m(1+r)}\frac{k}{k'} = 9.8$ generations. With these parameters, the efficient computation time is predicted to be 267 Mflops, yielding a predicted speedup of 2.64.

7.5.5 Computational results

A total of 50 trials were run for both the naive and efficient discretizations. As mentioned earlier, the naive and efficient computation times are expected to be 704 and 267 Mflops, respectively, with a speedup of 2.64. Tables 5 and 6 show the predicted and actual generations, function evaluations and computation time until convergence.

Table 5: Predicted and actual computation time values for groundwater experiments using a naive discretization.

	Generations	Time (Mflops)	Function evaluations
Predicted:	31.2	704	7050
Actual:	32.5	733	7350

Although the fitness contributions from the four zones were weighed, the transmissivity values converged at slightly different rates, with T_1 and T_3 converging before

Table 6: Predicted and actual computation time values for groundwater experiments using an efficient discretization.

	Generations	**Time (Mflops)**	**Function evaluations**
Predicted:	39.2	267	8860
Actual:	40.3	291	9110

T_2 and T_4, and the actual times for both implementations took slightly longer than was predicted as T_2 and T_4 finished converging. Despite this, the actual speedup was 2.52, close to the predicted speedup of 2.64.

8 Conclusions

This paper examines efficient GAs when fitness evaluations are estimated with implicit and explicit quadrature. Discretization scheduling efficiently matches the number of grid points to the accuracy required by the GA while not sacrificing solution quality. This paper introduces several important aspects for GA practitioners wishing to use discretization scheduling to run efficient genetic algorithms.

- Discretization scheduling efficiently uses computation time in problems with exponentially-scaled building blocks and a fitness function required quadrature.
- Discretization scheduling decreases the computation time by using coarse grained parameters in the early evaluations. This introduces error into the evaluations, which necessitates a larger population and a longer convergence time and, thus, more function evaluations. Despite the increase in total evaluations, the overall computation time decreases.
- For one-dimensional problems, speedup attained from discretization scheduling is largely dependent on the variable length, *k*, and the ratio of apparent variance error introduced by changing the grid to fitness variance, *r*, and to a lesser degree, the ratio α/β. For higher dimensional problems, the dimensionality, *d*, and polynomial time relationships between grid size and computation time, *p*, also affect the speedup. The population size, *N*, and the number of subfunctions, *m*, do not play a role in speedup.
- When discretization scheduling is used, a solution can be found in less time than using a constant grid without sacrificing solution quality.

Computational efficiency can be additionally improved by using other methods to decrease computation time, including parallelization, hybridization, evaluation relaxation and time utilization. The speedup in these four areas can be acquired independently and the total speedup of an efficient GA using the above four techniques is equal to the product of the individual speedups. Then the overall speedup S can be improved to

$$S = S_p S_h S_{er} S_{tu}.$$

where S_p is the speedup attained by running in parallel, S_h is the speedup from hybridization, S_{er} is the speedup from evaluation relaxation, and S_{tu} is the speedup attained by time utilization. Using discretization scheduling improves the time utilization.

There are several possible research directions that can be developed from this research. One of the assumptions is that the string is represented by binary bits. A theoretical approach for *k*-ary bits may prove useful for certain applications. Additionally, many real-world applications may contain equally salient or 0-1 variables in addition to other, exponentially-scaled variables. An example is groundwater modeling in which 0-1 variables are used to determine if a pump is on or off. An investigation to whether discretization scheduling could handle such variables would be useful for introducing efficiency to other types of applications using GAs. Finally, many of the efficient real-world applications consider additional heuristics to reduce the evaluation time. These are problem-specific ways to reduce the computation time, unlike the more general formulas developed in this paper. It would be meaningful to observe the speedup when using the heuristics with discretization scheduling together.

Acknowledgments

The first author's work was supported in part by the National Science Foundation (DMI-0114499) and the Air Force Office of Scientific Research (FA9550-04-1-0110).

The second author's work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-00-1-0163, and F49620-03-1-0129. Research funding for this work was also provided by the Technology Research Center (TRECC), a program of the University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by the Office of Naval Research under grant N00014-01-10175. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

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

References

Aizawa, A. N. and Wah, B. (1993). Dynamic control of genetic algorithms in a noisy enviroment. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 48–55.

Aizawa, A. N. and Wah, B. (1994). Scheduling of genetic algorithms in a noisy enviroment. *Evolutionary Computation*, 2(2):97–122.

Albert, L. A. (2001). Efficient genetic algorithms using discretization scheduling. Master's thesis, University of Illinois, Urbana, IL.

Albert, L. A. and Goldberg, D. E. (2000). The effect of numerical integration on the solution quality of a genetic algorithm. *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 15–21.

Albert, L. A. and Goldberg, D. E. (2001). Efficient evaluation relaxation under integrated fitness functions. *Intelligent Engineering Systems through Artificial Neural Networks*, 11:165–170.

Dahlquist, G. (1974). *Numerical methods*. Prentice Hall, Englewood Cliffs, NJ.

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

Eby, D., Averill, R. C., Gelfand, B., Punch, W. F., Mathews, O., and Goodman, E. D. (1997). An injection island for flywheel design optimization. In Zimmerman, H.-J., editor, *Eufit '97 – 5th European Congress on Intelligent Techniques and Soft Computing*, pages 687–691, Aachen, Germany. Verlag Mainz, Wissenschaftsverlag.

El-Beltagy, M., Nair, P., and Keane, A. (1999). Metamodeling techniques for evolutionary optimization of expensive problems: Promises and limitations. *Proceedings of Genetic and Evolutionary Computation Conference*, pages 196–203.

Fernández, F., Tomassini, M., Punch, W. F., and Sánchez, J. M. (2000). Experimental study of multipopulation parallel genetic algorithms. East Lansing, MI: Michigan State University.

Fitzpatrick, J. M. and Grefenstette, J. J. (1988). Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120.

Goldberg, D. E. and Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. *Proceedings of the Second International Conference On Genetic Algorithms*, pages 1–8.

Grefenstette, J. J. and Fitzpatrick, J. M. (1985). Genetic search with approximate function evaluations. In Grefenstette, J. J., editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 112–120, Hillsdale, NJ. Lawrence Erlbaum Associates.

Gwo, J., Hoffman, F. M., and Hargrove, W. W. (2000). Mechanistic-based genetic algorithm search on a Beowulf cluster. In *Proceedings of the High Performance Computing 2000 (HPC2000) Conference*, Washington, D. C.

Harrouni, K. E., Ouazar, D., Walters, G. A., and Cheng, A. H. (1996). Groundwater optimization and parameter estimation by genetic algorithm and dual reciprocity boundary element method. *Engineering Analysis with Boundary Elements*, 18(4):287–296.

Jin, Y., Olhofer, M., and Sendhoff, B. (2000). On evolutionary optimization with approximate fitness functions. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 786–793.

Jin, Y., Olhofer, M., and Sendhoff, B. (2001). Managing approximate models in evolutionary algorithms design optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 592–599, Seoul, Korea.

Karpouzos, D. K., Delay, F., Katsifarakis, K. L., and de Marsily, G. (2001). A multipopulation genetic algorithm to solve the inverse problem in hydrology. *Water Resources Research*, 37(9):2291–2302.

Kimura, M. (1964). *Diffusion models in population genetics*, volume 2 of *Supplementary review series in applied probability*. Methuen, London.

Kogiso, N., Watson, L. T., Gürdal, Z., and Haftka, R. T. (1993). Genetic algorithms with local improvement for composite laminate design. *Structures and Controls Optimization*, 38:13–28.

Le Riche, R. and Haftka, R. T. (1993). Optimization of laminate stacking for buckling load maximization by genetic algorihm. *AIAA Journal*, 31(5):951–956.

Lin, S. C., Punch, W. F., and Goodman, E. D. (1994). Coarse-grain parallel genetic algorithms: Categorization and new approach. *Sixth IEEE Parallel Distributed Processing*, pages 28–37.

Lobo, F. G., Goldberg, D. E., and Pelikan, M. (2000). Time complexity of genetic algorithms on exponentially scaled problems. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 151–158.

Miller, B. L. (1997). *Noise, Sampling, and Efficient Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL.

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

Miller, B. L. and Goldberg, D. E. (1996b). Genetic algorithms, tournament selection, and the varying effects of noise. *Complex Systems*, 9(3):193–212.

Miller, B. L. and Goldberg, D. E. (1996c). Optimal sampling for genetic algorithms. *Proceedings of the Artificial Neural Networks in Engineering*, pages 291–297.

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

Parmee, I. C., Vekeria, H., and Bilchev, G. (1997). The role of evolutionary and adaptive search during whole system, constrained and detailed design optimization. *Engineering Optimization*, 29:151–176.

Punch, W. F., Averill, R. C., Goodman, E. D., Lin, S. C., Ding, Y., and Yip, Y. C. (1994). Optimal design of laminated composite structures using coarse-grain parallel genetic algorithms. *Computing Systems in Engineering*, 5(4-6):415–423.

Ratle, A. (1998). Accelerating the convergence of evolutionary algorithms by fitness landscape approximations. *Parallel Problem Solving from Nature, Volume V*, pages 87–96.

Rothlauf, F. (2001). *Towards a theory of representations for genetic and evolutionary algorithms: Development of basic concepts and its application to binary and tree representations.* PhD thesis, University of Bayreuth, Germany.

Schraudolph, N. and Belew, R. (1992). Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21.

Shaefer, C. G. (1987). The ARGOT strategy: Adaptive representation genetic optimizer technique. *Proceedings of the Second International Conference on Genetic Algorithms*, pages 50–55.

Sun, N.-Z. (1994). *Inverse problems in groundwater modeling.* Kluwer Academic Publishers, Dordrecht.

Thierens, D., Goldberg, D. E., and Pereira, A. G. (1998). Domino convergence, drift and the temporal-salience structure of problems. *The 1998 IEEE Conference on Evolutionary Computation Proceedings*, pages 535–540.

von Wolfersdorf, J., Achermann, E., and Weigand, B. (1997). Shape optimization of cooling channels using genetic algorithms. *Transactions of the ASME*, 119:380–386.