Efficient Discretization Scheduling in Multiple Dimensions

Laura A. Albert
Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign
1206 W. Green St.
Urbana, IL 61801
(217) 244-0292
lalbert@uiuc.edu

David E. Goldberg
Department of General Engineering
University of Illinois at Urbana-Champaign
104 S. Mathews St.
Urbana, IL 61801
(217) 333-2346
deg@uiuc.edu

Abstract

There is a tradeoff between speed and accuracy in fitness evaluations when various discretization sizes are used to estimate the fitness. This paper introduces discretization scheduling, which varies the size of the discretization within the GA, and compares this method to using a constant discretization. It will be shown that when scheduling the discretization, less computation time is used without sacrificing solution quality. Fitness functions whose cost and accuracy vary because of discretization errors from numerical integration are considered, and the speedup achieved from using efficient discretizations is predicted and shown empirically.

1 Introduction

In recent years, advances in the field of genetic algorithms (GAs) have allowed an increasing number of complex real-world problems in optimization, search, scheduling, and machine learning to be solved with GAs. One obstacle to the use of GAs in commercial applications is the evaluation expense. As 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 the good part of a year to complete. Some of these applications with expensive function evaluations include computational fluid dynamics, structural optimization, and environmental applications. In many cases, coarse-grained parameters can be used to decrease the computation time by introducing 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 goal of this paper 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 must be obtained, and a model must be 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: For multidimensional problems, 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? This paper is organized as follows. First, we go over background, including decomposition of GAs, a literature review and error analysis of numerical integration. Next, we introduce the concept of discretization and discuss its components: convergence time, population sizing, and discretization considerations. Naive and efficient discretizations are then defined. Two-dimensional experiments are run to show the computational savings. Finally, this paper is concluded.

2 Background

In this paper, it is assumed that in the design of competent GAs—GAs that solve hard problems quickly, reliably, and accurately—can be decomposed into subproblems, and that each subproblem can be analyzed separately. The principles of the sixfold decomposition of GAs (Goldberg, Deb, & Clark, 1992) are

1. Know what the GA processes: building blocks (BBs)

2. Ensure there is an adequate initial supply of BBs
3. Ensure that necessary BBs are expected to grow
4. Ensure that BB-decisions are well made
5. Solve problem of bounded BB-difficulty
6. Ensure that BBs can be properly mixed

We are particularly interested in the third and fourth items, ensuring that necessary building blocks are expected to grow and that building block decisions are well-made. Efficient techniques that introduce error to the fitness evaluation may not ensure that the necessary building blocks grow if the decisions are made poorly. That is, error introduced by efficient techniques cannot be so large that the GA cannot choose correctly between two distinct, competing individuals in most cases, or the necessary building blocks will be lost from the population.

2.1 Literature Review

Since using discretization in a fitness function approximates the true fitness, it is critical to understand the effects of error on GA performance. In this section, we review previous research efforts that analyze and explain the effects of error on GAs. Although the type of error that we analyze is due to discretization and searching on a grid, research efforts that investigate other types of error, such as sampling error, can help to shed light on general effects of error in GA performance.

Early studies on the effects of evaluation error on GA performance considered evaluation error that was mainly due to variance or randomness. These studies used sampling to estimate the fitness and ran the GA in a bounded computation time. The tradeoff for these problems is between the time to make each fitness evaluation and the total number of evaluations made. Either more, inaccurate evaluations or fewer, accurate evaluations are made. In addition, each implementation has different convergence time and population requirements. Optimal sampling in GAs was studied by (Fitzpatrick & Grefenstette, 1988; Miller & Goldberg, 1996; Aizawa & Wah, 1994).

Other types of research considered industrial applications of GAs, with expensive fitness evaluations. One approach was to create an approximate model to the fitness landscape with statistical techniques, such as neural networks (El-Beltagy, Nair, & Keane, 1999; Jin, Olhofer, & Sendhoff, 2000). However, many of these types of approaches are particularly prone to converging to a sub-optimal solution because they search in the space as defined by the statistical technique, not the original problem. Others considered heuristics to improve GA performance for certain types of problems (Le Riche & Haftka, 1993; von Wolfersdorf, Achermann, & Weigand, 1997; Kogiso, Watson, Gürdal, & Haftka, 1993), but these approaches are strictly empirical and cannot be applied to other types of problems. Finally, the injection island GA refined the model in a stepwise process by running multiple grids for large engineering applications in parallel (Lin, Punch, & Goodman, 1994; Punch, Averill, Goodman, Lin, Ding, & Yip, 1994). Intuitively, the authors understood that some good building blocks could be generated inexpensively with a coarse grid spacing, so they ran several different resolution grids in parallel, each of which was run on a different set of processors. They then injected the best individuals into the higher resolution grids.

Albert and Goldberg (2000) empirically examined the tradeoffs between more and less accurate models that use numerical integration. When using a constant grid with a deterministic fitness function, bias error is added to the evaluations. They found that for a bounded computation time, an optimal number of grid points exists that can maximize solution quality. This research provided the initial motivation for discretization scheduling because the solution quality with this method may not be that which is desired.

Albert and Goldberg (2001) examined the amount of solution quality limited by the computation time available under one-dimensional integrated fitness functions. The objective was to minimize the computation time such that a given solution quality is acquired. 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.

2.2 Error Analysis

It is assumed that any numerical integration scheme used to estimate the fitness can be written of the form:

$$I(f) = \int_{a2}^{b2} \int_{a1}^{b1} f(x,y)dxdy \approx \sum_{i=1}^{n1} \sum_{j=1}^{n2} w_{ij} f(x_i, y_j)$$

where the integral of the function f taken at the points x_i and y_j is taken from $a1$ to $b1$ in the x-dimension and from $a2$ to $b2$ in the y-dimension, n_1 and n_2 are the total number of grid points in the x- and y-dimensions,

and w_{ij} 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 is $\frac{b1-a1}{n1}$ or $\frac{b2-a2}{n2}$. 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. That is, it is assumed that there are no additional costs to calculate the weights w_{ij}.

3 Discretization Scheduling

This section focuses on discretization scheduling and its components, and particularly the effects of the error that is introduced when using a coarse grid. The topics critical to discretization scheduling are apparent variance error, convergence time and population sizing.

The concept of discretization scheduling is to use more than one discretization size within the GA. The choice of the number of grid points n is traditionally fixed within the GA. This choice of 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. This is inspired by domino convergence (see section 3.2), which 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 at the beginning of the run, but in practice, this number may need to be larger.

Since convergence time tells us 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.

In the remainder of this section, we further elaborate on discretization scheduling. This section begins with the deriving of the computational requirements for GAs with fitness functions that consider numerical integration. However, much of the theory can be generalized to other, similar types of fitness functions.

The fitness function considered is an sum of m two-dimensional integrals. The experiments have the following form where f is the fitness function:

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

It is assumed that *g(x,y)* is continuous and differentiable.

3.1 Apparent Variance Error

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—the GA will take more generations to converge—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. It can be defined as

$$r = \frac{\sigma^2_{E,t}}{\sigma^2_{F,t}} \qquad (2)$$

where $\sigma^2_{E,t}$ is the error variance and $\sigma^2_{F,t}$ is the fitness variance, each at generation t. In addition, both the error and fitness variances decrease with time and are approximately zero at convergence. For the computational experiments in the following section, it is empirically shown that r is approximately constant during the GA when an efficient discretization is used.

3.2 Convergence Time

Convergence time is crucial for understanding how GAs perform and computation time can be used efficiently. In this section, 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 displayed. The convergence times are based on selection intensity adapted from population genetics (Kimura, 1964).

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 problems also converge in this way. Thierens, Goldberg, and Pereira (1998) 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 for binary integer subcodes using pairwise tournament selection can be expressed as

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

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 (Thierens, Goldberg, & Pereira, 1998). The convergence time for the entire GA can be found when considering that $k = l$ in the above equation. This equation implies that it takes the same amount of time to converge to each building block, no matter what its fitness contribution. When the whole string converges, λ is equal to the string length l.

The convergence times are different when the fitness function is composed of a sum of identical subfunctions, when the fitness can be written as:

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

The convergence time are developed in (Albert, 2001) and the derivations are not printed here. When evaluation error is not present, the convergence time can be written as:

$$t_{conv1} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi}}\right]}k = c_t\sqrt{m}k \tag{4}$$

where $c_t \approx \ln 2\sqrt{3\pi} = 2.13$, and the total convergence time increases by a factor of $\sqrt{m}$. The convergence rate is $O(\sqrt{m}k)$ for these types of functions.

When evaluation error is present, as mentioned in section 3.1, the convergence time is elongated. The convergence time changes to

$$t_{conv2} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m\pi(1+r)}}\right]}k = c_t\sqrt{m(1+r)}\cdot k \tag{5}$$

where, again, $c_t \approx \ln 2\sqrt{3\pi}$. The convergence rate is $O(\sqrt{m(1+r)}k)$ for these types of functions. This equation collapses to equation 4 when accurate fitness evaluations are used. It is assumed that r is constant or is approximately constant during the GA. It should be noted that it takes $c_k = c_t\sqrt{1+m}$ generations to converge to each bit in each subfunction.

The ratio of the convergence times can be determined from taking the ratio of equation 4 to equation 5. The ratio can thus be written as

$$\frac{t_{conv1}}{t_{conv2}} = \frac{c_t\sqrt{m}k}{c_t\sqrt{m(1+r)}k} = \frac{1}{\sqrt{1+r}} \tag{6}$$

Since $r \geq 0$, the above ration is always less than or equal to 1. That is, a GA with an accurate fitness function is on average expected to converge faster than its error-prone counterpart.

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 & Segrest, 1987):

$$t_{drift} \approx 1.4N \tag{7}$$

In this equation, N is the population size. For any application, it is important to ensure that drift will not occur until well after the population is expected to converge.

3.3 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—linearly. That is, a single, exponentially-scaled variable is expected to converge as $O(l)$ and has also been empirically observed that the population size must vary as $O(l)$ as well (Lobo, Goldberg, & Pelikan, 2000).

Recently, 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, Rothauf found that the population size varies as $O(k\sqrt{m})$. However, the model by Rothauf does not take into account building-block mixing, and the resulting population sizes given by this model are inadequate for our purposes.

In section 3.2 , 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 $O(k\sqrt{m(1+r)})$ because the GA converges at the slower rate of $O(k\sqrt{m(1+r)})$. From the observation of Lobo, Goldberg, and Pelikan (2000) and the theoretical work of Rothlauf (2001), it can be inferred that the convergence time and population size varies in the same way for exponentially-scaled problems.

The population sizes for our experiments are written in the form

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

where N_n is the naive population size and N_e is the efficient size. The constant c_N is the same in both equations and can be found empirically when using a naive discretization. The population size when using the efficient discretization can be determined once r is known.

3.4 Putting it All Together

A naive discretization assumes that the discretization is constant throughout the duration of the GA. This can be contrasted with an efficient discretization, in which the discretization varies within the GA (Albert, 2001). The computation time for a GA with a multiple integral can be modeled as

$$T = (\alpha + \beta n_1 n_2 ... n_d)GN_n.$$

where α is the overhead per individual per generation, β is the time to calculate one sample, n_i is the number of samples in the ith dimension, $G = t_{conv}$ is the total number of generations to convergence, and N is the population size. The term $(\alpha + \beta n_1 n_2 ... n_d)$ is the cost of each function evaluation. When the number of grid points in any dimension is identical, and $t_{conv} = c_t\sqrt{m}k$ is substituted for G, the above equation can be simplified to

$$T_n = (\alpha + \beta n^d)c_t k\sqrt{m} \cdot N_n. \quad (8)$$

where d is the number of dimensions, and N_n is the naive population size.

It is assumed that the discretization is scheduled in multiple dimensions by starting with two grid points in each dimension. Since the convergence time models tells us which bit the GA is converging to, it is also known how many grid points are used in each dimension. That is, when the GA is converging to the first bit, $2^1 = 2$ grid points are needed to converge to the correct first bit. It will take c_k generations to converge to the first bit, and the GA will start to converge to the second bit, when it will require $2^2 = 4$ grid points in each dimension. The GA will proceed in this way until it converges to the final kth bit when 2^k grid points are needed. Hence, the number of grid points in each dimension must be doubled in every dimension every c_k generations.

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 direction.

For an arbitrary amount of dimensions, d, the efficient time, as described above, can be written as

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

The above equation can be rewritten after substituting $N_e = N_n\sqrt{1+r}$ and $c_k = c_t\sqrt{m(1+r)}$:

$$T_e = \sum_{i=1}^{k}(\alpha + \beta \cdot 2^{d \cdot i})c_t\sqrt{m}N_n(1+r). \quad (9)$$

The speedup can be written in terms of the naive and efficient computation times:

$$S = \frac{T_n}{T_e} = \frac{(\alpha + \beta n^d)c_t\sqrt{m}k \cdot N_n}{\sum_{i=1}^{k}(\alpha + \beta \cdot 2^{d \cdot i})c_t\sqrt{m} \cdot N_n(1+r)} \quad (10)$$

Equation 10 can be simplified and rewritten knowing that $\sum_{i=1}^{k} 2^i = 2^{k+1} - 2$ and $n = 2^k$ as

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

4 Experiments

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

$$f = \sum_{i=1}^{3}\int_0^{x_i}\int_0^{y_i}\frac{\partial^2 g_i(x_i, y_i)}{\partial x_i \partial y_i}\partial y_i \partial x_i \quad (11)$$

$$g(x, y) = (e^{Ax}\cos(Bx))(e^{Ay}\cos(By)) \quad (12)$$

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 direction. However, 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 direction 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. This number is found by starting with 2 grid points and determining the largest difference between the location of the approximated optima and the actual optima. If this value is larger than the discretization size, h, then the number of grid points in increased. For this problem, the smallest discretization size such that the location of the approximated optima is within h of the true optima is 5, and 8 is the smallest power of two greater than 5.

4.1 Apparent error variance

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. The value of the variance is approximately 0.80 and it is roughly constant during the GA. Figure 1 shows how r varies within the GA. This value of r is used to predict the convergence time and the population size.

4.2 Convergence time analysis

The convergence time equation must be modified to take into account the three subfunctions and two variables in each subfunction. For any given fitness function (Muhlenbein & Schlierkamp-Voosen, 1993) give the following equation to describe how the fitness mean varies from one generation to another:

$$\mu_{t+1} - \mu = \sigma_{F,t} I \tag{13}$$

and from (Albert & Goldberg, 2001), when the fitness function is a sum of m identical subfunctions, the above equation can be rewritten as

$$m(\mu_{t+1} - \mu) = \sqrt{(}\sigma_{F,t} I)$$

If the subfunction can be written as $g(x, y) = h(x) \cdot h(y)$, then the equation 13 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:

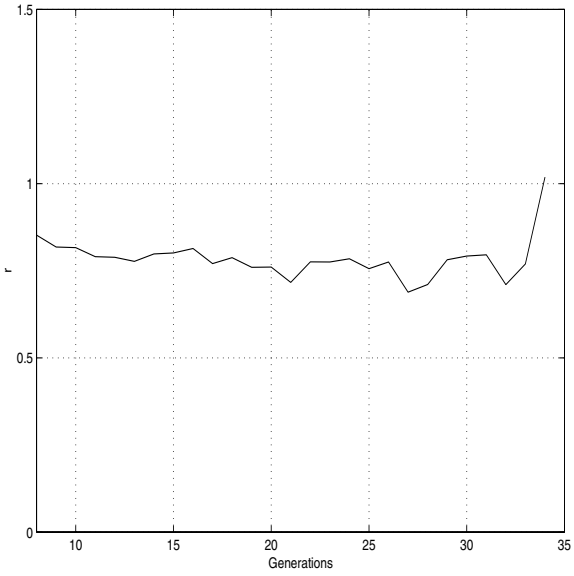

Figure 1: The value of r for the idealized two-dimensional experiments is roughly constant during the GA, and its value centers at 0.80

.

$$m'(\mu_{t+1}(\lambda) - \mu_t(\lambda)) = \sqrt{m'}\sigma_t(\lambda) I \tag{14}$$

The value $m' = 9$ is used in the convergence and population sizing equations. If binary tournament selection is considered, then the convergence times are

$$t_{conv1} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m'\pi}}\right]} k = c_t \sqrt{m'} k$$

$$t_{conv2} = \frac{-\ln 2}{\ln\left[1 - \frac{1}{\sqrt{3m'\pi(1+r)}}\right]} k = c_t \sqrt{m'(1+r)} \cdot k$$

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.

4.3 GA parameters

The population size was empirically determined by the following equations:

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

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

Experiments indicate that $c_N = 8.89$ for the population to be adequately sized, which results in a population size of 160 individuals for the naive experiments.

The population size 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. Thus, 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 $1/N$ for this problem. Pairwise tournament selection, one-point crossover and uniform mutation are the operators used for this problem.

4.4 Time budgeting and speedup

The expected computation time for the naive case can be found when α and β are known. Here, the two dimensional case is considered, $d = 2$. 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 subfunction, 64 grid points are needed in each dimension. Equation 8 can predict the time needed for the naive implementation and the following equation predicts the time for the efficient implementation.

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 and predicted number of function evaluations is 10,580. 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 directions.

4.5 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 1 and 2 show the generations, time, and number of fitness evaluations for the naive and efficient implementations.

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

	Gens	**Time**	**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 2 shows the converged gene position for each subfunction for the naive and efficient implementation. Like in the one-dimensional experiments, the experimental gene position follows very closely to the predicted gene position.

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

	Gens	**Time**	**Evaluations**
Predicted:	49.2	1.59	10,750
Actual:	50.0	1.69	10,750

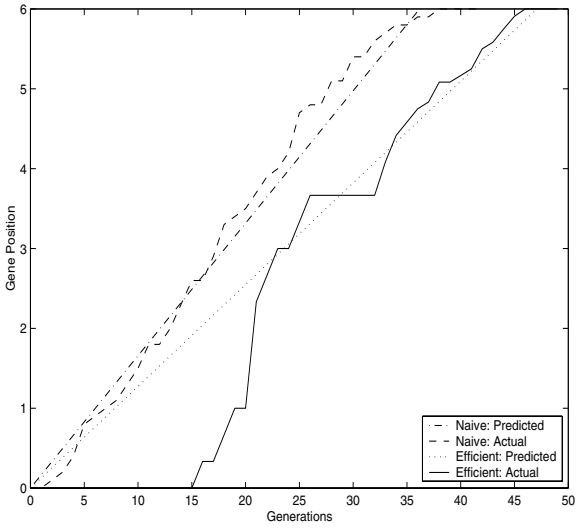

Figure 2: Predicted and actual gene position for idealized two-dimensional experiments

5 Conclusions

Discretization scheduling allows a GA to efficiently use computation time on a single processor by changing the grid spacing during the GA. This is particularly useful for fitness functions that use a discrete system to approximate a continuous system and that are computationally expensive. We show that, theoretically, it is possible to schedule the grid in multiple dimensions such that computation time is decreased while solution quality is not sacrificed. Empirically, we show that these relationships hold for a two-dimension nu-

merical integration problem.

Acknowledgments

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. 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 National Science Foundation, or the U.S. Government.

References

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

Albert, L., & Goldberg, D. (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*, 15–21.

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

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

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

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

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

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

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

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

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

Le Riche, R., & 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., & Goodman, E. D. (1994). Coarse-grain parallel genetic algorithms: Categorization and new approach. *Sixth IEEE Parallel Distributed Processing*, 28–37.

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

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

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

Punch, W. F., Averill, R. C., Goodman, E. D., Lin, S. C., Ding, Y., & 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.

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.* Doctoral dissertation, University of Bayreuth, Germany.

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

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