Scheduling of Genetic Algorithms in a Noisy Environment

Akiko N. Aizawa
National Center for Science Information Systems
3-29-1 Otsuka, Bunkyo-ku
Tokyo 112, JAPAN
akiko@nacsis.ac.jp

Benjamin W. Wah*
Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street, Urbana, IL 61801, U.S.A.
wah@manip.crhc.uiuc.edu

Abstract

In this paper, we develop new methods for adjusting configuration parameters of genetic algorithms operating in a noisy environment. Such methods are related to the scheduling of resources for tests performed in genetic algorithms. Assuming that the population size is given, we address two problems related to the design of efficient scheduling algorithms specifically important in noisy environments. First, we study the *duration-scheduling problem* that is related to setting dynamically the duration of each generation. Second, we study the *sample-allocation problem* that entails the adaptive determination of the number of evaluations taken from each candidate in a generation. In our approach, we model the search process as a statistical selection process and derive equations useful for these problems. Our results show that our adaptive procedures improve the performance of genetic algorithms over that of commonly used static ones.

Keywords

genetic algorithms, noisy environment, search scheduling, statistics theory, sample allocation

1. Introduction

Genetic algorithms (Holland, 1975; Goldberg, 1989) provide robust yet efficient procedures for guiding searches even in the absence of domain knowledge. Examples of such applications are found in such areas as parameter optimization in complex systems, and heuristic learning or strategy acquisition through experience (De Jong, 1988; Booker, Goldberg & Holland, 1990; Wah, 1992; Goldberg, 1989).

In these *knowledge-lean* domains, it is difficult to express an application in a well-defined model and analyze its behavior. A feasible way for finding a good solution is to measure the performance of candidate solutions through actual experimentation or simulation. This is particularly difficult when the application environment is *noisy*. For example, consider a parameter-optimization problem in which search points correspond to different control parameter values; the objective of the search is to select a suitable parameter set through experimentation. Noise in the experiments can be caused by randomness in the underlying

* Research supported by National Science Foundation Grant MIP 92-18715 and Joint Services Electronics Program Contract N00014-90-J-1270.
Preliminary results of part of this paper have appeared in references (Aizawa & Wah, 1993, 1994).

environment, uncertainty in the control signals, or variation of conditions used in the experiments. Since such noise is generally expressed by a stochastic process, we refer to the evaluation process as *sampling* and the observed value as a *sample* from the candidate being evaluated.

In a noisy environment, at least a few evaluations are needed to estimate the performance of a candidate solution accurately. Generally, better accuracy on the performance value can be obtained when more tests are performed on each candidate. On the other hand, the more candidates we examine, the greater the probability of encountering better ones. Since the total testing time is finite, there is a trade-off between the accuracy of estimation and the number of candidate solutions examined in a search. This trade-off, although important, is difficult in knowledge-lean application domains where domain knowledge relating quality and cost is missing (Wah, 1992). A good sampling (*scheduling*) strategy is required to obtain a better solution candidate within a finite amount of time.

This paper focuses on the scheduling of computational resources in genetic algorithms operating in a noisy environment. A scheduling algorithm is a triplet $\langle M, T, N \rangle$ where

M : population size in each generation

T: duration of each generation

N: number of evaluations for each candidate

In our terminology, M is the number of candidates maintained in one generation; T, the total number of samples assigned to each generation; and N, the number of samples taken from each candidate. In a standard implementation of genetic algorithms, it is implicitly assumed that these three parameters are constant and do not change between or within generations. Assuming that unit time is needed to sample a candidate, the relationship among these three parameters is:

$$T = M \times N. \tag{1}$$

The most common approach to scheduling is to determine the best population size M based on schema analysis, assuming that duration T is given (Fitzpatrick & Grefenstette, 1988; Goldberg, Deb, & Clark, 1991).

In this paper, we assume that M is given and that T and N can vary between or within generations. When T and N vary between generations, generations are sampled with different amounts of time, and candidates in different generations are sampled with different numbers of tests. When N varies within a generation, candidates in a generation are sampled different numbers of times. Let t_k be the total number of samples tested in the kth generation, and $n_{i,k}$ be the sample size of the ith candidate in the kth generation. The relationship corresponding to Equation 1 is:

$$t_k = \sum_{i=1}^{M} n_{i,k}, \qquad (k = 1, 2, \ldots). \tag{2}$$

This paper investigates two types of scheduling problems. The first problem, the *duration-scheduling problem*, focuses on the effect of varying T and N between generations. The sample size N is common for all candidates in each generation (that is, $n_{1,k} = n_{2,k} = \cdots = n_{M,k}$). The second problem, the *sample-allocation problem*, focuses on within-generation scheduling in which N may be different for different candidates in each generation, while T is kept constant (that is, $t_1 = t_2 = \cdots$).

Our approach to these scheduling problems is based on statistical observations that the behavior of genetic algorithms can generally be expressed by a statistical model whose parameters can be estimated from sampled values. Based on the statistical model, we explore ways to determine adaptively the schedules used in genetic algorithms.

This paper is organized as follows. Section 2 presents a *static scheduling policy* with constant T and N and shows the advantage of using *dynamic policies*. Section 3 presents a statistical model of the generate-and-test process used in genetic algorithms. We also show methods for collecting statistics when a genetic algorithm is executed. In Section 4, we introduce the Bayesian posterior distribution used in our scheduling policies. This distribution is derived using the statistical model defined in Section 3. We also present analytical results to compare the Bayesian and non-Bayesian methods. In Section 5, we study the duration-scheduling problem, and in Section 6 we derive decision equations for the sample-allocation problem. Section 7 shows the improvement in performance of our dynamic scheduling policies over the conventional static ones. Finally, conclusions are drawn in Section 8.

2. Description of the Problem

2.1 Scheduling Resources in Conventional Genetic Algorithms

It is known that population size has significant influence on the performance of genetic algorithms. This is explained by the schema theory as follows. The population size determines the diversity of schemata evaluated during the search. Holland (1975) showed that for most practical values of N and L, a population of size N contains $O(N^3)$ schemata. Goldberg, Deb, and Clark (1991) pointed out that this does not necessarily imply that larger population sizes are better. They showed that the optimal population size is determined by a trade-off between the number of existing schemata in a population and the rate of schema processing. When the population size is too small, genetic algorithms tend to converge prematurely, whereas their convergence is often too slow when the population size is large.

Such analysis is also applicable to cases in noisy environments (Fitzpatrick & Grefenstette, 1988; Grefenstette, Ramsey, & Schultz, 1990). The selection of a suitable population size in a noisy environment is often discussed using the concept of *variance of fitness* (Goldberg & Rudnick, 1991; Goldberg, Deb, & Clark, 1991).

Consider a class of candidates represented by schema H. The performance values of these candidates are different, even if they contain the common schema H. The *variance of fitness* of schema H is the variance of the performance values of candidates that contain schema H. In the case of noisy environments, the variance value is simply increased by a certain amount to take the environmental effect into consideration.

An example of such population sizing in a noisy environment is shown by Fitzpatrick and Grefenstette (1988). They studied the best population size M for a genetic algorithm, assuming that the duration time T is given. Their analysis is based on the schema theory. Let r be the number of candidates with hyperplane H, where $r \propto M$. Further, let ${\sigma_S}^2$ be the variance of sample means of the r candidates; σ_H be the variance of the true performance of the r candidates; $\sigma^2(c_i)$ be the variance of samples from candidate c_i; and $\langle \sigma^2(c_i) \rangle_H$ be the average sample variance of the r candidates with hyperplane H. Applying a simple statistical analysis, they showed the following equation (Fitzpatrick & Grefenstette, 1988);

$$ {\sigma_S}^2 = \frac{1}{r} {\sigma_H}^2 + \frac{1}{rN} \langle \sigma^2(c_i) \rangle_H . \tag{3} $$

Since duration time T is given, rN in Equation 3 is constant according to Equation 1. In short, Equation 3 shows that when T is given (and the cost of generating new candidates is

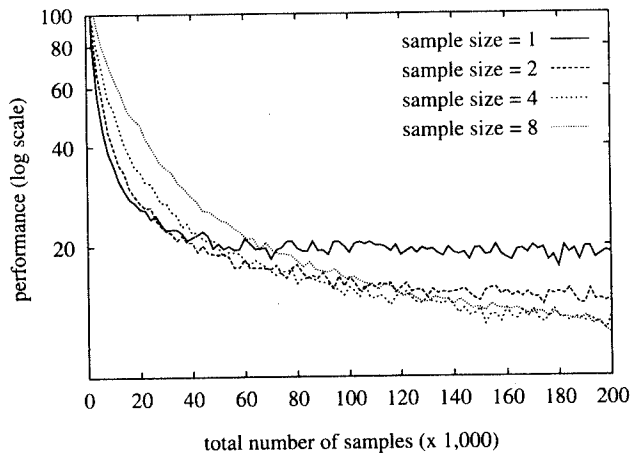


Figure 1. Behavior of static scheduling policies.

negligible), a genetic algorithm will perform better with smaller sample sizes (it will be the best when $N = 1$).[1]

2.2 Problems with Static Scheduling Policies

Although there exist a number of studies on genetic algorithms operating in noisy environments, most analytical studies deal with static scheduling policies where M, T, and N are chosen beforehand and remain unchanged throughout the operation of the genetic algorithm. Contrary to past approaches, we study in this paper the case in which population size M is given, and we evaluate the effect of T and N on performance. Such an assumption is important in practice because possible values for M are often restricted by the available computational resources, the requirement on convergence, and the characteristics of the search space. When only M is given, Equation 3 is not useful because it only indicates that taking more samples within one generation is better.

Assuming M is given, we first approach the static scheduling problem in two steps. First, we assume that samples are allocated equally among candidates. Hence, T is uniquely determined by N using Equation 1 for a given M. Next, we examine the optimal N (and, therefore, optimal T) by enumerating possible values of N ($= 1, 2, \cdots$). Figure 1 shows the effect of changing N (and accordingly T) for $M = 100$ in a minimization problem. The conditions for the experiments are the same as those of Fitzpatrick & Grefenstette (1988), except that we have used a sample variance of $\sigma = 64.0$ instead of $\sigma = 2.0$ in their study. Here, we have used a relatively large variance to demonstrate its effect on genetic algorithms with a fixed scheduling policy. This effect is similar for smaller variances, only that it takes more time to see the effect.

1 In order to make our analysis consistent, we have modified the symbols used by Fitzpatrick and Grefenstette (1988). Their original equation is $\sigma_S{}^2 = \frac{1}{r}\sigma^2 + \frac{1}{rn}\langle\sigma^2(x_i)\rangle_H$.

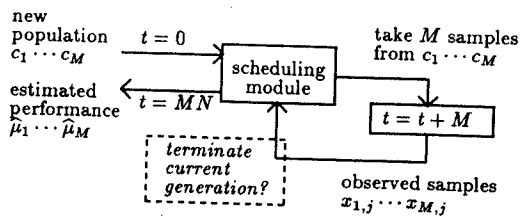


Figure 2. Duration-scheduling problem.

It is clear from Figure 1 that there is no optimal static scheduling policy that performs the best consistently even for the same problem, and that the best value of N depends on the total execution time. In general, smaller sample sizes are better in early generations, and larger ones are better in later generations. For example, in Figure 1, $N = 1$ performs the best in the beginning but performs the worst at the end.

In short, policies with fixed T and N (called *static policies*) do not work well when M is predetermined. Adjusting N for a specified deadline is not practical because there is no systematic way to know the exact future behavior of a genetic algorithm. The problem is more complex when deadlines are soft, involving trade-offs between improvement in solution quality and additional overhead of sampling.

2.3 Definition of the Scheduling Problems

Based on the above observation, we focus in this paper on methods for determining N, T, or both dynamically during the execution of a genetic algorithm. As described in Section 1, we classify the scheduling problems into two types.

- *Duration-scheduling problem* (Figure 2). This problem entails methods for determining when to stop the current generation. Candidates in the same generation are assumed to be sampled equally. This is also called *between-generation scheduling*.
- *Sample-allocation problem* (Figure 3). This problem involves methods for allocating t_0 samples among M candidates, where t_0 is given. This is also called *within-generation scheduling*.

Duration scheduling addresses the problem of premature/slow convergence of genetic algorithms due to an inappropriate duration size (i.e., the estimation of the performance of candidates is unnecessarily accurate or inaccurate) and is applicable when there are plenty of computational resources. In contrast, sample allocation is important when each evaluation (or sample drawn) is costly. Consequently, it is essential to select tests carefully in order to have the maximum benefit on the test results and to use the results to adjust the sample size dynamically. Note that these two problems do not occur concurrently; hence, we do not attempt to evaluate their combined effects in the same genetic algorithm.

These two problems are basically decision-making problems. For the duration-scheduling problem, the decision is to determine when to stop the current generation; for the sample-allocation problem, the decision is to determine the candidate to sample next.

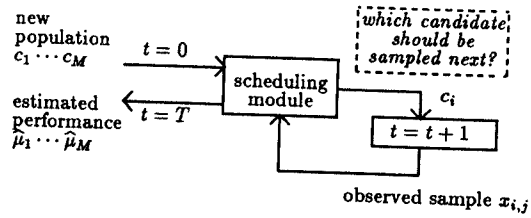


Figure 3. Sample-allocation problem.

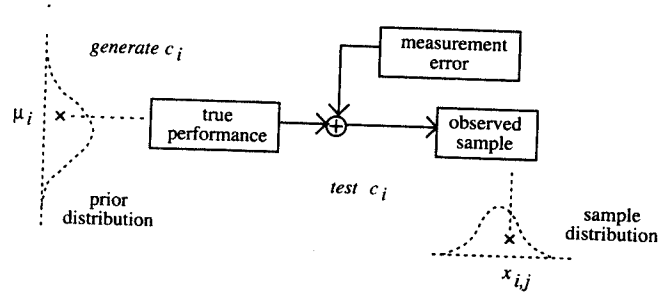


Figure 4. Statistical model of the generate-and-test process.

3. Statistical Model and Assumptions

3.1 Model of Generate-and-Test Process

To address the scheduling problems defined in Section 2.3, we use a statistical model to represent the generate-and-test process in genetic algorithms. In our model shown in Figure 4, generation and testing procedures are modeled as independent statistical processes.

3.1.1 Statistical Model of the Testing Process In the testing phase, the performance of candidates in the current generation is evaluated. In our model, testing a candidate is equivalent to picking a sample from a distribution.

Let $\mu_{i(k)}$ be the "true" performance of candidate c_i in the kth generation. Note that the "true" performance is unknown due to noise in the environment. Here, we assume the evaluation noise to be common for all candidates and be invariant in time.

Denoting the jth sample from c_i as $x_{i,j(k)}$, and the evaluation noise for $x_{i,j(k)}$ as $e_{i,j(k)}$, we have

$$x_{i,j(k)} = \mu_{i(k)} + e_{i,j(k)}. \tag{4}$$

The distribution of $x_{i,j(k)}$ is called the *sample distribution* of c_i and is denoted as $f(x \mid \mu_i)$.

3.1.2 Statistical Model of the Generation Process In the generation process, candidates in the kth generation are created by applying randomized genetic operators to candidates in the $(k-1)$st generation. Since all candidates are assumed to have the same statistical

properties, creating a new candidate can be interpreted as picking a sample from a set having a given distribution.

Let $E_{i(k)}$ be the "noise" in the generation process caused by the probabilistic nature of genetic operators. The performance of the ith candidate in the kth generation is then expressed as

$$\mu_{i(k)} = \mu_{0(k)} + E_{i(k)}, \tag{5}$$

where $\mu_{0(k)}$ is the average of expected performance of candidates in the kth generation. The distribution of μ_i is called the *prior distribution* and is denoted as $h(\mu)$. It expresses the expected performance of an arbitrarily chosen candidate in generation k.

When both the generation and testing processes are considered, $x_{i,j(k)}$ of the sample drawn can be expressed as

$$x_{i,j(k)} = \mu_{0(k)} + E_{i(k)} + e_{i,j(k)}. \tag{6}$$

In the rest of this paper, we omit the suffix (k) when it is obvious.

3.2 Normality Assumptions

In our statistical model, we assume normal distributions for both the sample and prior distributions.

First, we assume that the evaluation noise is Gaussian. Denoting the variance of the evaluation noise as σ^2, the sample distribution is Gaussian $N(\mu_{i(k)}, \sigma^2)$, where

$$f(x \mid \mu_i) = \frac{1}{2\pi\sigma} e^{-\frac{(x-\mu_i)^2}{\sigma^2}}. \tag{7}$$

The Gaussian-noise assumption is so widely accepted in numerous fields that we take it for granted in this paper.

Second, we assume that the prior distribution is expressed as a normal distribution. As defined before, $\mu_{0(k)}$ is the (expected) average performance of the kth generation. Let $\sigma_{0(k)}{}^2$ be the (expected) deviation of the performance of individual candidates in the same generation. The prior distribution of the kth generation is then a normal distribution $N(\mu_{0(k)}, \sigma_{0(k)}{}^2)$, where

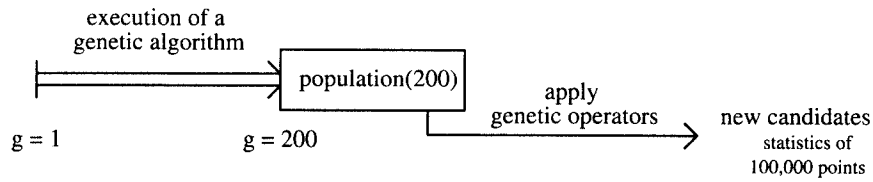
$$h(\mu) = \frac{1}{2\pi\sigma_0} e^{-\frac{(\mu-\mu_0)^2}{\sigma_0{}^2}}. \tag{8}$$

The building-block hypothesis in genetic algorithms suggests the validity of the normality assumption. From the Law of Large Numbers, we can assume that the overall distribution is normal since the performance of a candidate is the sum of effects of many small components (schemata).

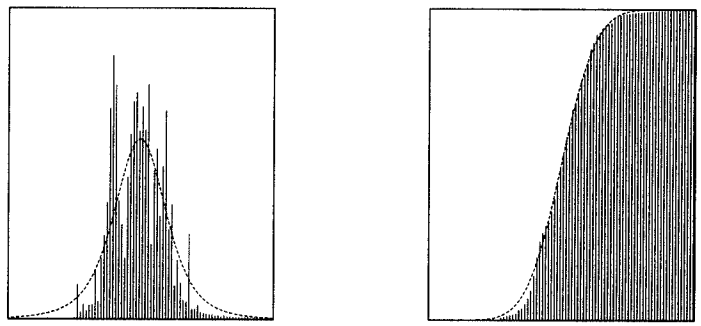
As the sum of two normal distributions is also a normal distribution, Equation 6 corresponds to the so-called two-factor analysis, where $x_{i,j}$ is expressed as a normal distribution $N(\mu_0, \sigma_0{}^2 + \sigma^2)$.

Precisely speaking, the exact form of the prior distribution is not normal. The distribution of the kth generation should be uniquely determined by the binary codes of the $(k-1)$st generation and the genetic operators applied. Since such an "exact" form can only be verified empirically, we sometimes refer to such distribution as *empirical distributions* of the kth generation (denoted as $h_0(\mu)$).

Figure 5 shows an example of a comparison between the "normal" and the "exact" prior distributions. In this example, a genetic algorithm is first executed 200 generations. At



(a) Outline of the experiment



(b) Probability density function (c) Cumulative density function

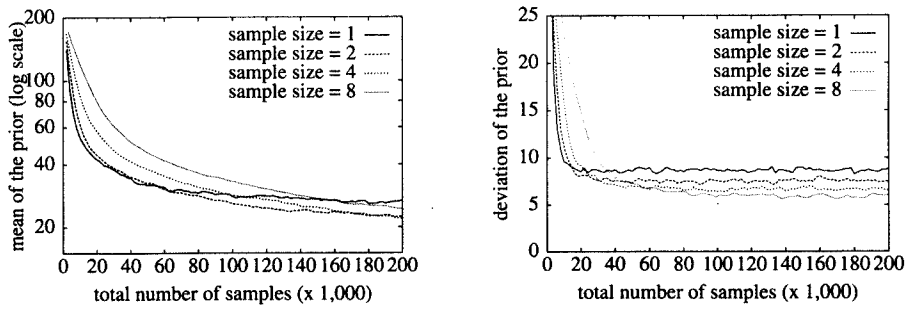
Figure 5. Example of an empirical distribution.

the end of the 200th generation, genetic operators are applied repeatedly to create 100,000 new candidates (Figure 5(a)). The empirical probability density function (PDF) and the cumulative distribution function (CDF) are drawn for these 100,000 points. The mean and the standard deviation of the distribution are calculated, and the corresponding normal distribution is derived (Figure 5(b) and 5(c)). The empirical distribution is then approximated by a normal distribution shown as dotted lines in Figure 5(b) and 5(c). The effects of approximation are discussed in the next section.

3.3 Estimation of Statistical Parameters

So far, we have assumed two distribution functions: the sample distribution $f(x \mid \mu) \sim N(\mu, \sigma^2)$, and the prior distribution $h(\mu) \sim N(\mu_0, {\sigma_0}^2)$. This means that our model uses three parameters that should be estimated from the samples obtained:

- σ^2: the variance of the noise
- $\mu_{0(k)}$: the mean of the μ_i's in the kth generation
- ${\sigma_{0(k)}}^2$: the variance of the μ_i's in the kth generation



(a) Transitions of $\mu_{0(k)}$ (b) Transitions of $\sigma_{0(k)}$

Figure 6. Transitions of $\mu_{0(k)}$ and $\sigma_{0(k)}$ between generations.

First, we estimate σ^2 using the following equation for estimating common variance (Crow, Davis, & Maxfield, 1960):

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{M} \left((n_i - 1)s_i^2\right)}{\left(\sum_{i=1}^{M} n_i\right) - M}, \qquad \text{where } (n_i - 1)s_i^2 = \sum_{j=1}^{n_i} \left(x_{i,j} - \bar{x}_i\right)^2. \tag{9}$$

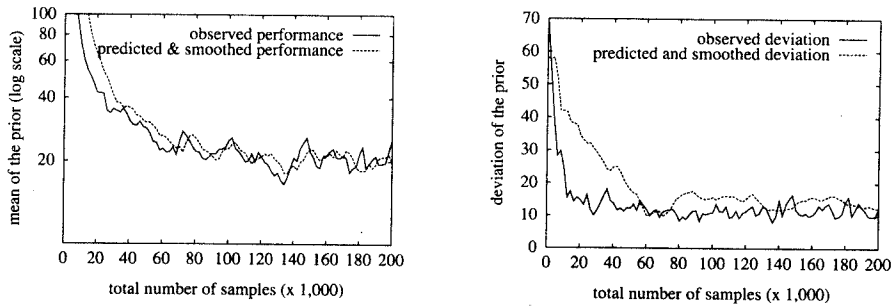
When samples are obtained for the current generation, the estimation of μ_0 and ${\sigma_0}^2$ is the same as in two-factor analysis (Crow, Davis, & Maxfield, 1960). When $n_1 = \cdots = n_M = n$, the distribution of the $\bar{x}_i$'s is normal with mean μ_0 and variance $\left({\sigma_0}^2 + \frac{\sigma^2}{n}\right)$. Therefore, $\mu_0(k)$ and ${\sigma_0}^2(k)$ can be estimated as:

$$\hat{\mu}_{0(k)} = \frac{1}{M} \sum_{i=1}^{M} \bar{x}_i, \tag{10}$$

$$\hat{\sigma}^2_{0(k)} = \frac{1}{M(M-1)} \left(M \sum_{i=0}^{M} (\bar{x}_i)^2 - \left(\sum_{i=0}^{M} \bar{x}_i \right)^2 \right) - \frac{\hat{\sigma}^2}{n}. \tag{11}$$

The initial values of these three parameters are obtained through pre-sampling, where a negligible number of samples are taken. (For example, four samples can be taken from each candidate in the first generation.) We assume that σ does not change during the execution; however, this is not true for μ_0 and σ_0. Figure 6 shows the changes in $\mu_{0(k)}$ and ${\sigma_{0(k)}}^2$ during the execution of a genetic algorithm with different sample sizes. In general, $\mu_{0(k)}$ gradually improves as $\sigma_{0(k)}$ becomes smaller.

The discussion above shows that we should update the estimations of μ_0 and σ_0 in each generation and predict these values for the next generation using the values obtained in previous generations. Figure 7 shows an example of parameter estimation using Equations 9,



(a) Prediction of $\mu_{0(k)}$ (b) Prediction of $\sigma_{0(k)}$

Figure 7. Predicted and observed $\mu_{0(k)}$ and $\sigma_{0(k)}$ between generations.

10, and 11. Although the empirical values fluctuate due to sampling error, the predicted curve, smoothed using a moving average, is smooth. (The current moving average was computed by adding the current predicted value and 0.7 of the previous moving average.) Although we can apply regression analysis for the prediction, we compute the moving average for simplicity in this paper. Note that in this example, only one sample is taken from each candidate; if more samples are drawn, the estimate would be improved.

As existing samples are used for these estimations, no additional pre-sampling is needed after the first generation.

As is demonstrated later in this paper, the estimated statistical parameters we have developed in this section are very useful in leading to good schedules. The use of exact statistical parameters may not be necessary and may be computationally intractable.

4. Statistical Inference and Analysis

4.1 Bayesian Posterior Distribution

In the testing process, the prior and sample distributions described in Section 3 are used to estimate the statistical parameters of individual candidates. With the normality assumption, the prior distribution represents *a prior* belief or knowledge about the performance of the candidate *before* it is actually tested. In contrast, the belief or knowledge about the value of μ_i *after* we actually test candidate c_i is called the *posterior distribution*. The posterior distribution is a conditional distribution calculated from the prior distribution and sampled values.

Using the Bayes theorem, the *posterior distribution* of μ_i is derived as follows (Lloyd, 1984). Assume that n_i samples are drawn from candidate c_i, and that the sample mean is $\bar{x}_i$. Let $h(\mu)$ be the prior distribution and $h_i^*(\mu \mid \bar{x}_i, n_i)$ be the posterior distribution of c_i. The distribution of $\bar{x}_i$ is expressed as

$$\bar{f}(\bar{x}_{i,n_i} \mid \mu) \sim N\left(\mu, \frac{\sigma^2}{n_i}\right). \tag{12}$$

Using the Bayes theorem, the posterior distribution is derived as

$$h_i^*(\mu \mid \bar{x}_{i,n_i}) = \frac{\bar{f}(\bar{x}_{i,n_i} \mid \mu)h(\mu)}{\int_{-\infty}^{\infty} \bar{f}(\bar{x}_{i,n_i} \mid \mu)h(\mu)d\mu}. \tag{13}$$

When squared-error loss is applied, the best estimator for μ_i (denoted as μ_i^*) is the expected value of the posterior (conditional) distribution defined in Equation 13.

$$\mu_i^* = \int_{-\infty}^{\infty} \mu\, h_i^*(\mu \mid \bar{x}_{i,n_i})d\mu. \tag{14}$$

When we have no knowledge about the distribution of μ_i's, we can use the *non-informative prior* which is defined as $h(\mu) = (const)$ for all μ. From Equation 13, the posterior distribution in this case is equal to the sample distribution. Also, we can use the empirical distribution $h_0(\mu)$ as a prior and calculate h_i^* in Equation 13 and μ_i^* in Equation 14 by numerical integration. As we can measure the empirical distribution accurately, we can obtain a good estimation of all possible prior distributions.

When the prior distribution is normal, Equation 13 is also normal as shown below.

$$h_i^*(\mu \mid \bar{x}_i, n_i) \sim N\left(\frac{n_i\bar{x}_i + \alpha\mu_0}{n_i + \alpha}, \frac{\sigma^2}{n_i + \alpha}\right), \qquad \alpha = \frac{\sigma^2}{{\sigma_0}^2}. \tag{15}$$

We denote the mean and variance of the above normal distribution as μ_i^* and σ_i^{*2}, respectively. μ_i^* gives the best (Bayes) estimation with respect to μ_i, and σ_i^{*2} gives the expected estimation error (assuming the squared-error loss), where

$$\mu_i^* = \frac{n_i\bar{x}_i + \alpha\mu_0}{n_i + \alpha}, \qquad \sigma_i^{*2} = \frac{\sigma^2}{n_i + \alpha}. \tag{16}$$

When both the sample and prior distributions are normal, α_i is an indicator of the strength of our belief about the performance of c_i without actually sampling it. Figure 8 shows an example of the prior and posterior distributions computed using Equation 16. We assume that the average performance of candidate c_1 is 1.0 after taking two samples. When we have a strong belief that μ_1 is close to 0, the posterior distribution is still close to 0 (Figure 8(a)). In contrast, when we have a vague belief about μ_1, the posterior distribution is close to 1 using the same sample set (Figure 8(b)).

4.2 Analysis

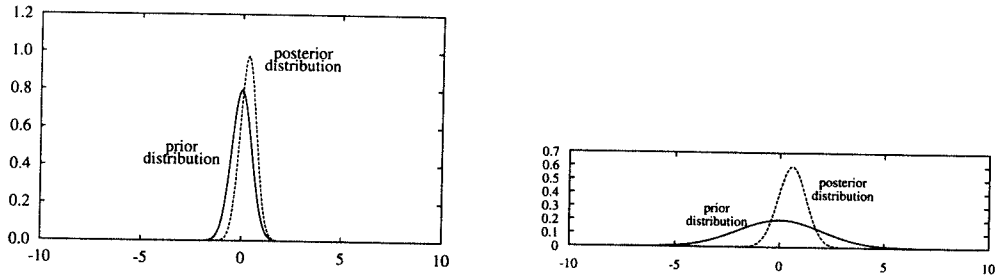
In this section we analyze the effect of applying prior distribution to estimate the performance of each candidate. The following three prior distributions are compared:

a. no prior

b. normal prior

c. exact (empirical) prior

As an empirical distribution, we use the one obtained in Section 3. As a normal prior, we use a normal distribution with the same mean and variance as the empirical distribution. Using these prior distributions, we calculate and depict the corresponding posterior distributions in Figure 9. Among the three posterior distributions shown in Figure 9, the one derived



(a) With strong prior belief (b) With vague prior belief

Figure 8. Examples of posterior distribution with strong and vague prior.

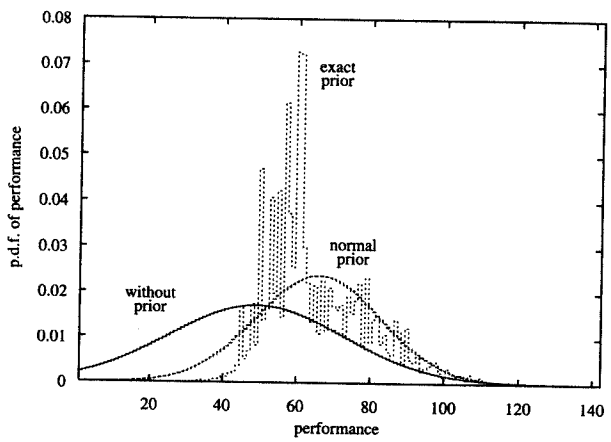


Figure 9. Comparison of the posterior distributions.

from the exact prior (case *c*) gives the ideal prediction. Comparing the other two cases *a* and *b* with the ideal case, we see that using the normal prior is more accurate both in terms of the mean and variance of the distribution.

To evaluate these different priors quantitatively, we calculate the expected (squared) error of the performance estimation for each distribution. Denoting the empirical distribution as h_0, the expected error can be expressed as

$$\int_{\mu=-\infty}^{\infty} \left[\int_{\bar{x}=-\infty}^{\infty} (\mu - \mu_i^*)^2 \bar{f}(\bar{x}_{i,n_i} \mid \mu)\, d\bar{x} \right] h_0(\mu)\, d\mu. \tag{17}$$

The inside integrand in Equation 17 shows the expected estimation error when the "true" performance is μ. μ^* is obtained using Equation 14, where $h(\mu)$ is an arbitrary prior distribution (our belief). It is integrated over the exact prior distribution $h_0(\mu)$.

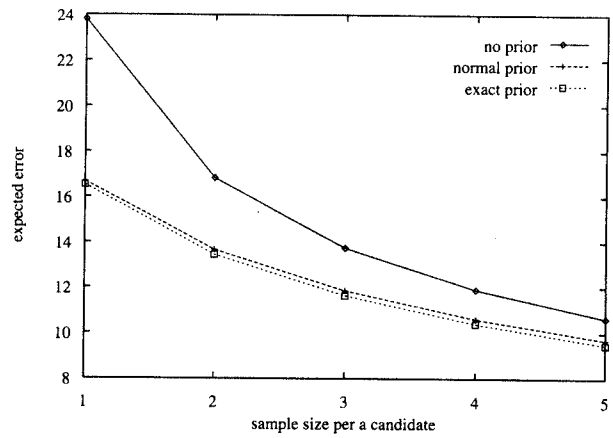


Figure 10. Effect of the prior distribution.

In Figure 10, we compare the error of these three estimations. We see that the normal prior is fairly close to the exact prior case. Note that the exact prior gives the best case.

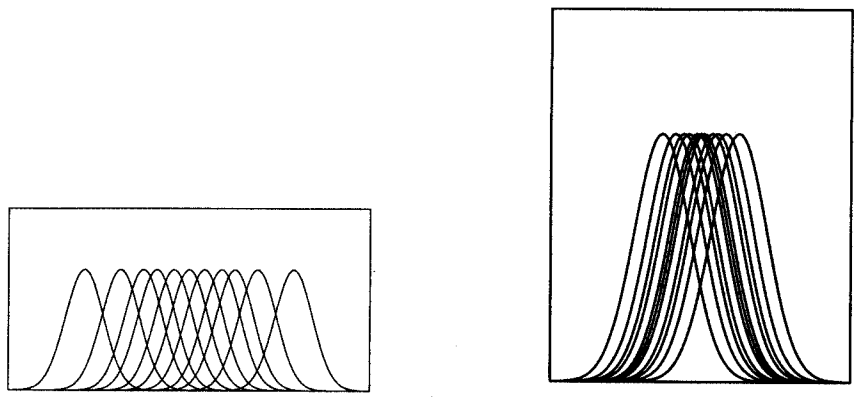
We note that the use of the prior distribution $h_i(\mu)$ is meaningful only when $h_i(m)$ is a good guess of the real value of m_i. As shown in Figure 8, the prior distribution with a sharp peak has much prominent effect on the calculation of the posterior distribution. Therefore, inaccurate strong belief (prejudice) can distort considerably the performance of the samples. This is not surprising, because the prior distribution is only our knowledge to enhance our evaluation, and inferencing with the wrong knowledge is often worse than inferencing with no knowledge. Because the values of μ_0 and σ_0 used in the actual genetic algorithm are only estimated, we examine the effectiveness of applying normal prior in our simulation study in Section 7.

5. Between-Generation Scheduling: Duration-Scheduling Problem

5.1 Objective

In executing a genetic algorithm, useful schemata are identified implicitly by selecting candidates with better sample means. Statistically, the difficulty of selection in the kth generation can be characterized by the *variance ratio* (denoted as $\beta(k)$), which is the ratio of the sample variance and the variance of the μ_i's; that is, $\beta(k) = \sigma^2/\sigma_{0(k)}{}^2 (= 1/\alpha)$. When $\beta(k)$ is small, the μ_i's are distributed "sparsely," and the selection of better candidates is easy. On the other hand, when $\beta(k)$ is large, then the μ_i's are distributed close to each other, and the selection is difficult. Figure 11 illustrates the cases with small and large $\beta(k)$.

The above discussion explains why static scheduling does not work well. Note that in Figure 6(a), the distribution of the μ_i's is "dense" at first and gradually becomes "sparse" as time progresses. Since the variance of the μ_i's varies between generations, no sample size N can be the best throughout all the generations. Hence, we can expect the overall performance to improve if we use smaller sample sizes in early generations and larger ones in later generations.



(a) Small $\beta(k)$ (b) Large $\beta(k)$

Figure 11. Selection problem with different variance ratios.

5.2 Equations for Scheduling

The above observation leads us to choose the following duration (or the number of samples drawn) in the kth generation in our scheduling strategy.

$$t_k = M \times \left(n_0 + \left\lceil \gamma \sum_{l=1}^{k-1} t_l \right\rceil \right). \tag{18}$$

In this case, $\sum_{l=1}^{k-1} t_l$ is the total number of samples observed so far. Equation 18 simply means that we increase the duration time at fixed intervals of time.

We also set the following conditions to guarantee that the candidate variance is within a certain range.

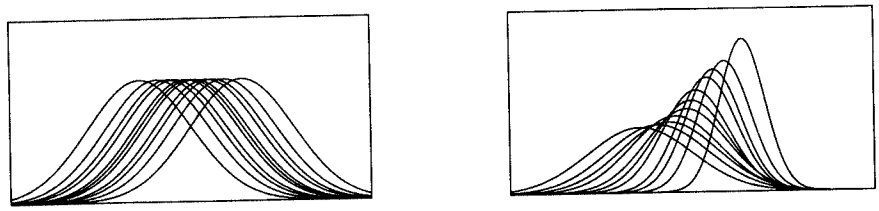
$$\delta_1 \leq \frac{\sigma/\sqrt{n}}{\sigma_{0(k)}} = \frac{\sqrt{\beta(k)}}{n} \leq \delta_2. \tag{19}$$

Equation 18 is only applied when the following conditions are satisfied. If $\sqrt{\beta(k)}/n < \delta_1$, then we consider the candidates to be distributed "sparsely" enough so that we can distinguish them from each other without further sampling. Thus, the current generation is terminated immediately. On the other hand, if $\sqrt{\beta(k)}/n > \delta_2$, then we consider the candidates to be distributed too "densely," and we need to take more samples, and the current generation is not terminated even if Equation 18 is satisfied.

In this paper, we choose γ heuristically as 5×10^{-3}, δ_1 as 1.0, and δ_2 as 4.0.

5.3 Scheduling Procedure

To summarize, our scheduling procedure for determining the duration of a generation consists of four steps.



(a) Equal n_i (b) Unequal n_i

Figure 12. Equal and unequal allocation of n_i's.

1. Determine the initial sample size n_0 using Equation 19. Here, the value of $\beta(k)$ is determined by the estimation of σ^2 and $\sigma_{0(k)}{}^2$ using Equations 9 and 11 shown in Section 3.3.
2. Take one sample from each of the candidates in the current generation.
3. If the current time t is less than t_k as defined in Equation 18, then go to Step 2.
4. Else terminate the current generation; generate new candidates for the next generation; and go to Step 1.

6. Within-Generation Scheduling: Sample-Allocation Problem

6.1 Objective

Our approach to this problem is based on decision theoretic methods (Ferguson, 1967; Ghosh & Sen, 1991). Our objective is to determine an efficient allocation $(n_1, \ldots, n_M)$ so that the expected value of a pre-defined loss function (or *risk*) is minimized. As a loss function, we use the estimation error (σ_i^{*2}) of the candidate selected. Our choice is motivated by the fact that candidates with better performance have larger probability of being selected for reproduction in the next generation. Hence, we can improve the performance of genetic algorithms by sampling more from better candidates (i.e., by decreasing the estimation error of these candidates), and by spending less time on inferior ones. (See Figure 12.)

6.2 Equations for Scheduling

Our scheduling policy tries to minimize the expected risk by determining adaptively the candidate to be sampled next. The expected risk is expressed as follows:

$$\overline{R} = \sum_{i=1}^{M} P_i^* \, \sigma_i^{*2}, \tag{20}$$

where P_i^* is the probability that candidate c_i is the best. We calculate P_i^* as:

$$P_i^* = \int_{-\infty}^{\infty} \left[\prod_{j \neq i} H_j^*(\mu \mid \overline{x}_j, n_j) \right] h_i^*(\mu \mid \overline{x}_i, n_i) d\mu, \tag{21}$$

where $H_j^*(\mu \mid \bar{x}_j, n_j) = \int_{-\infty}^{\mu} h_j^*(\mu \mid \bar{x}_j, n_j)d\mu$ is the CDF of h_i^*. Let ϕ and Φ be the PDF and CDF of the standard normal distribution, respectively. Using μ_i^* and σ_i^* defined in Equation 16, Equation 21 becomes

$$P_i^* = \int_{-\infty}^{\infty} \left[\prod_{j \neq i} \Phi\left(\frac{\mu - \mu_j^*}{\sigma_j^*} \right) \right] \phi\left(\frac{\mu - \mu_i^*}{\sigma_i^*} \right) d\mu.$$

Note that P_i^* could have been defined as the actual selection probability $f_i / \sum_{i=1}^{M} f_i$; however, we use Equation 21 because it is less dependent on the specific fitness function (especially the scaling factor) used in individual genetic algorithms. For ease of calculation, we make the simplifying assumption that P_i^* is independent of σ_i^{*2}; therefore, $\partial P_i^* / \partial n = 0$ for all i.

Now let $(n_1^+, \ldots, n_M^+)$ be the optimal (desired) allocation for Equation 21. By applying Lagrange multiplier λ, we obtain the following M equations:

$$\frac{\partial}{\partial n_i} \left[\sum_{i=1}^{M} P_i^* \sigma_i^{*2} - \lambda \sum_{i=1}^{M} n_i \right] = 0, \qquad i = 1 \cdots M. \tag{22}$$

Since $\frac{\partial P_i^*}{\partial n_i} = 0$, it immediately follows that:

$$P_i^* \frac{\partial \sigma_i^{*2}}{\partial n_i} - \lambda = 0, \qquad i = 1 \cdots M. \tag{23}$$

Therefore, $(n_1^+, \ldots, n_M^+)$ should be chosen in such a way that

$$-P_1^* \frac{\sigma^2}{(n_1^+ + \alpha)^2} = -P_2^* \frac{\sigma^2}{(n_2^+ + \alpha)^2} = \cdots = -P_M^* \frac{\sigma^2}{(n_M^+ + \alpha)^2}. \tag{24}$$

Equation 24 means that when the samples are optimally allocated, each term of the equation should be equal. Comparing the term $-P_i^* \frac{\sigma^2}{(n_i+\alpha)^2}$ for the current and the desired allocations, we obtain the feedback value for each i. Note that we can only increase n_i for samples drawn in the future; hence, the best sampling strategy is to select candidate i with the largest feedback value:

$$\textit{Select } c_i \textit{ such that} \quad P_i^* \frac{\sigma^2}{(n_i + \alpha)^2} - P_i^* \frac{\sigma^2}{(n_i^+ + \alpha)^2} = \max_{1 \le j \le M} \left[P_j^* \frac{\sigma^2}{(n_j + \alpha)^2} - P_j^* \frac{\sigma^2}{(n_j^+ + \alpha)^2} \right]$$

$$\iff P_i^* \frac{\sigma^2}{(n_i + \alpha)^2} = \max_{1 \le j \le M} \left[P_j^* \frac{\sigma^2}{(n_j + \alpha)^2} \right]. \tag{25}$$

Since the calculation of P_i^* requires complex numerical integration, further simplification is sometimes desirable to reduce its complexity. We interpret P_i^* to represent the importance of candidate c_i. Hence, if we substitute P_i^* by a reasonable weight that can be calculated more easily, then we can avoid the costly computation of P_i^*. As an empirical simplification, we define a heuristic risk function using heuristic weight w_i^*.

$$\bar{\bar{R}} = \sum_{i=1}^{M} w_i^* \sigma_i^{*2}. \tag{26}$$

For w_i^*, we consider only the probability that candidate c_i is better than the current best candidate (or second best, in case that c_i is the best). Let i be the permutation of indices such that $\mu_{(i)}^*$ is the ith estimated mean and $\mu_{(k)}$ is the largest. Then w_i^* is defined as:

$$w_i^* = \int_{-\infty}^{\infty} H_j^*(\mu \mid \bar{x}_j, n_j)\, h_i^*(\mu \mid \bar{x}_i, n_i) d\mu \qquad where\, j = \begin{cases} (k) & \text{for } i \neq (k) \\ (k-1) & \text{for } i = (k) \end{cases} . \tag{27}$$

The computational complexity is $o(k)$ for the simplified version as compared to $o(k^2)$ for the gradient-based sampling. For the normal-distribution case, we use μ_i^* and σ_i^* defined in Equation 16. w_i^* can then be simplified as:

$$w_i^* = \Phi\left(\frac{\mu_i^* - \mu_j^*}{\sqrt{\sigma_i^{*2} + \sigma_j^{*2}}}\right) \qquad where\, j = \begin{cases} (k) & \text{for } i \neq (k) \\ (k-1) & \text{for } i = (k) \end{cases} . \tag{28}$$

The scheduling policy based on the computation of the heuristic risk function in Equation 26 is the same as before; that is,

$$Select\ c_i\ such\ that\ w_i^* \frac{\sigma^2}{(n_i + \alpha)^2} = \max_{1 \leq j \leq M} \left[w_j^* \frac{\sigma^2}{(n_j + \alpha)^2} \right] . \tag{29}$$

6.3 Scheduling Procedure

To summarize, our scheduling procedure for sample allocation is as follows.

1. Sample once each of the M candidates.
2. Calculate σ_i^* and P_i^* (or w_i^* as an approximation) for each candidate.
3. Select the candidate with the largest feedback using Equation 25 (or Equation 29 as an approximation).
4. Sample the candidate selected in Step 3.
5. Repeat Steps 2 through 5 until $t = T$.

7. Experimental Results

7.1 Experimental Conditions

To illustrate the effectiveness of dynamic scheduling, we compare the performance of static policies with that of dynamic ones using simple GA test functions. In our simulation study, we assume that candidate c_i is represented by a binary code $\mathbf{z}_i$ ($= z_{i,1}, z_{i,2}, \ldots$) and the "true" evaluation value μ_i is determined by a test function F, such that $\mu_i = F(\mathbf{z}_i)$. As is expressed in Equation 4, we add a Gaussian noise with variance σ^2 when c_i is evaluated in a simulation.

In the next two subsections, we use two test functions. The first function F_1 is De Jong's unimodal function (De Jong, 1975).

$$F_1(\mathbf{z}_i) = \sum_{m=1}^{30} i\, z_{i,m}{}^4 , \quad (-1.28 < z_{i,m} \leq 1.28, z_{i,m}\ has\ 8\ bits). \tag{30}$$

The second function F_2 (Mühlenbein, Schomisch, & Born, 1991) is a highly multimodal function.

$$F_2(z_i) = 20 + \sum_{m=1}^{20} \left({z_{i,m}}^2 - \cos(2\pi z_{i,m}) \right), \quad (-5.12 < z_{i,m} \leq 5.12, z_{i,m} \textit{ has } 10 \textit{ bits}). \quad (31)$$

The size of the solution space is 2^{240} for F_1 and 2^{200} for F_2. Both functions have the global minima when all $z_{i,m} = 0$. Note that F_2 has 7^{20} local minima. When the z_i's are created randomly, the distribution of the μ_i's is almost normal: for F_1, $\mu_{0(0)} = -225$ and $\sigma_{0(0)} = 69$; and for F_2, $\mu_{0(0)} = -194$ and $\sigma_{0(0)} = 35$. The evaluation noise is set to be $\sigma = 1.0 \times \sigma_{0(0)}$.

For other simulation conditions, we use the standard genetic-algorithm parameters with population size of 100, crossover rate of 0.6, mutation rate of 0.001, and scaling-window size of 7.

Simulation results using six other test functions are shown in Section 7.3.

7.2 Performance of Policies Based on Static and Dynamic Duration Sizes

In our first experiments, we examine the effect of applying dynamic policies for the duration-scheduling problem. The total evaluation time is set to be $T_{total} = 200,000$ in order for us to compare the performance of different policies over a long time range.

Figure 13 shows the typical behavior observed for a dynamic policy. The x-axis shows the generation number, and the y-axis, the number of samples per candidate allocated in the generation. The figure shows that the duration size, and accordingly the number of samples per candidate, increases steadily in the long run, while the number of samples drawn in a particular generation may fluctuate. The number of samples drawn may increase when the population variance in a generation decreases. This happens because the relative magnitude of noise increases when the population variance decreases, and more samples have to be drawn in order to reduce the estimation error. The global trend in the increase in the number of samples drawn is given by constant γ in Equation 18, whereas the fluctuation is given by constraints δ_1 and δ_2 in Equation 19. (Recall that δ_1, δ_2, and γ are chosen to be 1.0, 4.0, and 5.0×10^{-3}, respectively.)

Table 1 shows the average computation overhead for a population of size 100 for both static and dynamic policies. There is no difference in the overhead between static and dynamic policies because dynamic policies only incur negligible overheads to compute posterior mean for each candidate and mean and variance of the current population at every fixed number of generations. We use t_α to denote the cost of generating a new candidate and t_β for the cost of sampling the population once. In the current implementation, t_α / t_β is 7.9 when the population size $M = 100$. Since this value is highly problem dependent, we assume that t_α / t_β are variables in the following experiments. We achieve this in the functions tested by increasing the overhead of each function evaluation using statements that perform no operation.

Figure 14 and Figure 15 show the effect of using static versus dynamic duration sizes. In these figures, the x-axis is the total test time normalized by t_β, and the y-axis is the performance of the current top candidate ("on-line" performance). Since we assume relatively small sampling cost, we set the generation gap to be 1.0, and assume that no sample information is carried from previous generations to future generations. The result shown is the average over 50 runs.

In Figure 14, we assume that the cost for creating a new population is negligible ($t_\alpha / t_\beta = 0.0$). Consequently, the x-axis corresponds to the total number of samples examined. For

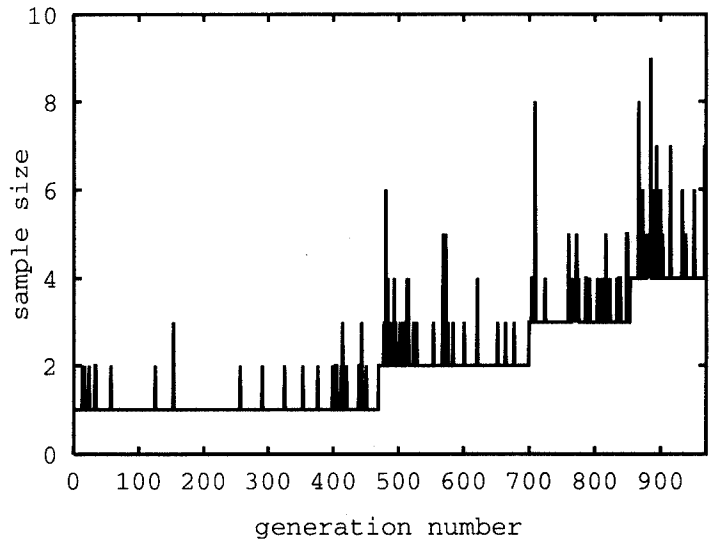


Figure 13. Typical behavior of a dynamic policy in duration scheduling.

Table 1. Computation overhead for the duration scheduling problems.

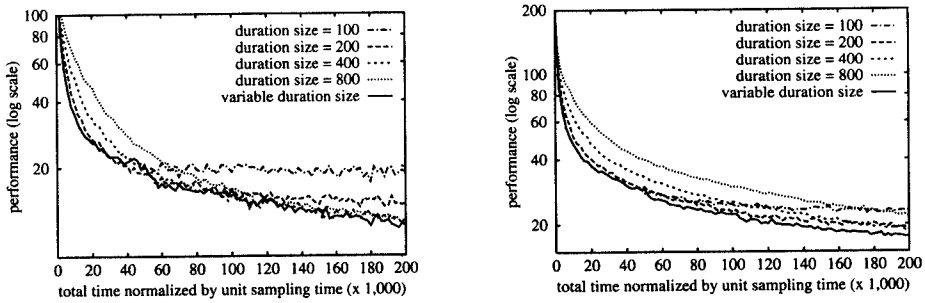
	cost for creating a population once (t_β)	cost for sampling a new population (t_α)
static policies	173 (msec)	22.0 (msec)
dynamic policies	173 (msec)	22.0 (msec)

both F_1 and F_2, the performance of our policy using dynamic duration sizes is at least as good as that of the best static policy at any point in time.

In Figure 15, we assume that the cost for creating a new population is greater than the cost for sampling it once (t_α/t_β = 3.0). In this case, scheduling using small sample sizes is better than that using large sample sizes. For example, when the normalized time is greater than 100 in Figure 14(b), the performance for duration size of 800 is generally worse than that for other duration sizes, whereas in Figure 15(b), the performance for duration size of 800 is better than that for duration size of 100.

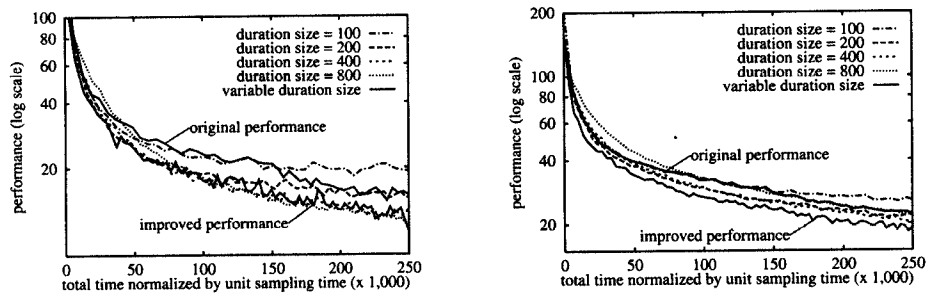
Figure 15 shows that the performance of the dynamic policy used in Figure 14 is worse (for the values of γ, δ_1 and δ_2) than that of static policies, assuming negligible generation cost. To overcome this problem we heuristically increase the sample size of the original dynamic policy by 2. The improved performance of the modified dynamic policy is also shown in Figure 15. The method for choosing the sample size is left for future study.

In short, we have shown that, regardless of the value of t_α/t_β, the search sometimes "gets stuck" when the variance of the population is small as compared to the variance of the evaluation noise. In this case, increasing the sample size (i.e., decreasing the evaluation noise) will bring the search out of the local minima. In our experiments, we have observed



(a) Test function F_1 (b) Test function F_2

Figure 14. Effect of variation in duration sizes when $t_\alpha / t_\beta = 0$.



(a) Test function F_1 (b) Test function F_2

Figure 15. Effect of variation in duration sizes when $t_\alpha / t_\beta = 3$.

relatively high values of t_α / t_β and have evaluated cases for a range of t_α / t_β values. In general, more costly evaluation functions may be used and will result in negligible generation cost ($t_\alpha / t_\beta = 0$).

7.3 Performance of Policies Based on Static and Dynamic Sample Sizes

We show in this section our experimental results in applying dynamic policies for the sample-allocation problems. Since we assume that the sampling cost is high for these problems, we use total evaluation time $T_{total} = 5{,}000$. Further, samples drawn in previous generations are carried over to future generations, which means that the sample mean and sample size of a candidate in the current generation are maintained when the candidate survives in the next generation.

Figure 16 shows the typical behavior observed for dynamic sample-allocation policies. The x-axis is the generation number, and the y-axis is the average number of samples allocated for the top five candidates (solid line) and the worst five candidates (dashed line). The

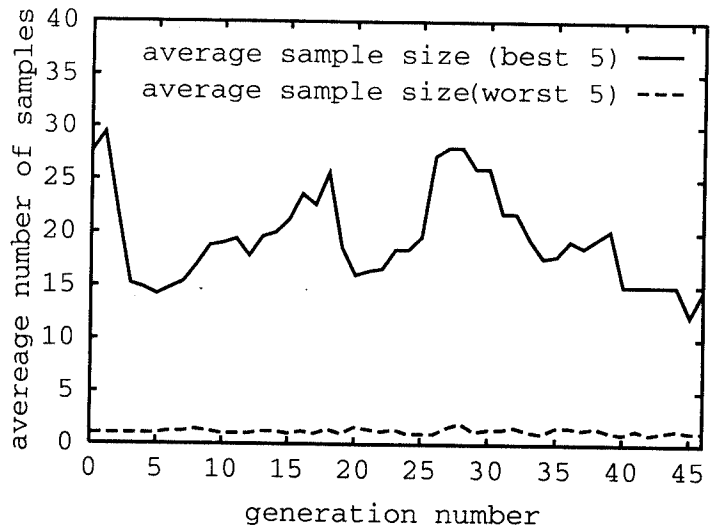


Figure 16. Typical behavior of dynamic policies in sample allocation.

Table 2. Computation overhead for the sample-allocation problems.

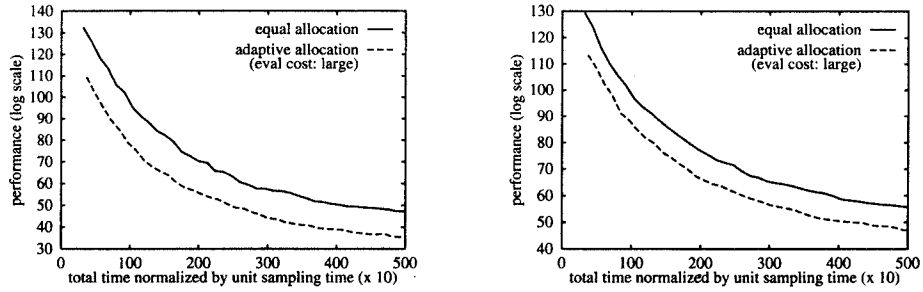
	cost for creating a new population (t_α)	cost for sampling a population once
static policies	173 (msec)	t_β = 22.0 (msec)
dynamic policies	173 (msec)	t'_β = 61.5 (msec)

numbers include samples carried over from previous generations. The figure shows that candidates with good performance are sampled repeatedly, whereas candidates with poor performance are examined mostly once. Note that the average trend of these curves is not increasing. This happens because candidates in a population are evaluated based on their relative performance.

Table 2 shows the average computation overhead for a population of size 100 for both static and dynamic policies. Since dynamic policies need extra computation time to determine the specific population to sample for each sample drawn (Equations 27 and 28), the overhead in this case is not negligible. For example, when the duration size is T = 200, the observed computation overhead of dynamic policies is 0.4 times more than that of static policies for the simple test functions studied (Equations 30 and 31).

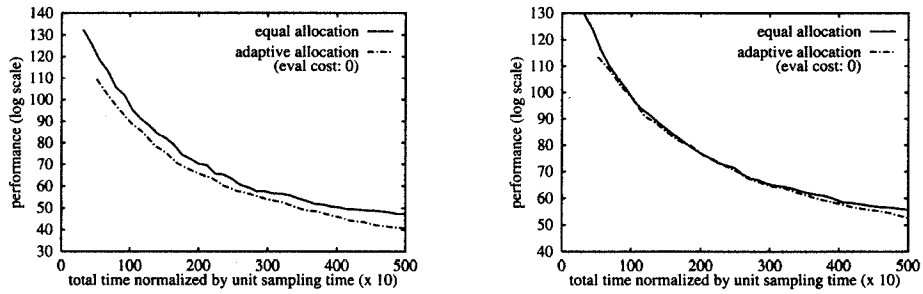
We denote the sampling cost of dynamic policies as t'_β. Again, this value is implementation dependent and we treat t_β and t'_β as variables. For the allocation problems studied, we do not consider the cost for generating a new population (t_α) since the duration time T is common for both static and dynamic policies.

Figures 17 and 18 show the performance results for static and dynamic sample-allocation policies. The x-axis is the total test time normalized by t_β and t'_β, respectively, and the y-axis



(a) Test function F_1 (b) Test function F_2

Figure 17. Effect of sample allocation when $t'_\beta \approx t_\beta$.



(a) Test function F_1 (b) Test function F_2

Figure 18. Effect of sample allocation when $t'_\beta = 3.0\ t_\beta$.

is the performance of the current top candidate. The generation gap is set to be 0.6, and the best 40% of candidates survive in the next generation. The results plotted are the average of 200 runs. For static allocation, we use a sample size of 2, which is the best within the time range.

In Figure 17, we assume that the additional computation cost for the dynamic policy is negligible ($t'_\beta \approx t_\beta$). For both F_1 and F_2, our dynamic policy consistently outperforms the static one.

In Figure 18, we assume that the additional computation cost for a dynamic policy is not negligible ($t'_\beta = 3.0\ t_\beta$). Here, our dynamic policy still outperforms the static one for both test functions.

Since the evaluation time in our example is only 220 μsec per candidate, it is very likely that the assumption $t'_\beta \approx t_\beta$ holds in most of the cases.

Table 3. Simulation conditions.

Simulation Conditions	Duration-Scheduling Problem	Sample-Allocation Problem
Test functions	$f_1 \sim f_6$	$f_1 \sim f_6$
Sample variance	$\sigma = 0.5\sigma_0 \sim 2.0\sigma_0$	$\sigma = 0.5\sigma_0 \sim 3.0\sigma_0$
Population size	$M = 30 \sim 100$	$M = 30 \sim 100$
Deadline	$T = 10,000 \sim 300,000$	$T = 2,000 \sim 10,000$

Table 4. Test functions.

$f_1(\mathbf{z}_i) = \sum_{m=1}^{20} (10\, z_{i,m}^4)$
$f_2(\mathbf{z}_i) = \sum_{m=1}^{20} [{z_{i,m}}^2 + (1 - \cos(2\pi z_{i,m}))]$
$f_3(\mathbf{z}_i) = \sum_{m=1}^{20} [z_{i,m}^2 + \mid z_{i,m} \mid (1 - \cos(2\pi z_{i,m}))]$
$f_4(\mathbf{z}_i) = \sum_{m=1}^{20} z_{i,m} \times (10\, z_{i,m}^4)$
$f_5(\mathbf{z}_i) = \sum_{m=1}^{20} z_{i,m} \times [{z_{i,m}}^2 + (1 - \cos(2\pi z_{i,m}))]$
$f_6(\mathbf{z}_i) = \sum_{m=1}^{20} z_{i,m} \times [z_{i,m}^2 + \mid z_{i,m} \mid (1 - \cos(2\pi z_{i,m}))]$

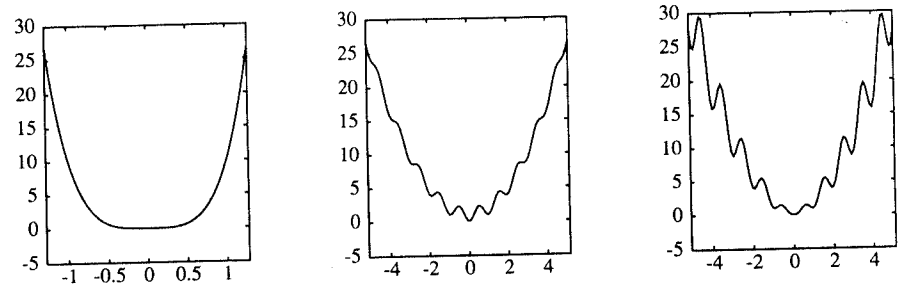
7.4 Overall Evaluation

In this section, we show the performance of the static and dynamic scheduling policies for additional test functions under a variety of configuration parameters. Table 3 shows conditions we have used in our simulations. It also defines six test functions $f_1 \sim f_6$ (Table 4) for $\mathbf{z}_i = z_{i,1}, z_{i,2}, \ldots, z_{i,20}$, where $z_{i,m}$ is represented in 10 bits and $-512 < z_{i,m} < 512$. All functions have the global minimum at $z_{i,m} \equiv 0$. f_1 and f_4 are unimodal; f_2 and f_5 are multimodal with large local minima around the optimal points; f_3 and f_6 are also multimodal but with smaller local minima around the optimal points. For $f_1 \sim f_3$, $z_{i,m}$'s are weighted equally, while for $f_4 \sim f_6$, $z_{i,m}$'s are weighted differently. Figure 19 depicts the change in performance values along one dimension ($z_{i,m}$) of three of these functions.

For each run of the simulation, one set of conditions is selected randomly from Table 3, and the performance is measured under the same condition for both:

1. genetic algorithm with static scheduling
2. genetic algorithm with dynamic scheduling

For static policies, we used sample sizes $N = 1, 2, 4, 8$. For each scheduling policy, we repeated the simulations five times and reported the average performance. We also show for comparison the "off-line" performance, which is obtained by taking an additional 10



(a) $f(x) = 10x^4$ (b) $f(x) = x^2 + [1 - \cos(2\pi x)]$ (c) $f(x) = x^2 + |x| [1 - \cos(2\pi x)]$

Figure 19. Behavior of test functions f_1, f_2, and f_3 used in our experiments.

samples from the top 20 candidates after the last generation is completed. We denote the performance values as $Y_{s(1)}$, $Y_{s(2)}$, $Y_{s(4)}$, $Y_{s(8)}$ for the static policies and Y_d for the dynamic policy, respectively. For each policy we report its performance normalized with respect to Y_d: $(Y_{s(1)}/Y_d)$, $(Y_{s(2)}/Y_d)$, $(Y_{s(4)}/Y_d)$, and $(Y_{s(8)}/Y_d)$. Note that from the definition of the test functions, the optimal performance is equal to zero for all cases.

Table 5 shows the result obtained for the duration-scheduling problems, assuming that the generation cost is negligible ($t_\alpha/t_\beta = 0$). The first column is the normalized performance averaged over 250 simulation runs under different simulation conditions. Since the problems defined in Table 4 are minimization problems, "$Y_{s(N)}/Y_d > 1.0$" means that the dynamic policy performs better than the static policy with sample size N. The second column is the number of simulation runs (out of 250) in which the static policy performs better than the dynamic policy.

On the average, the dynamic policy performs better than the static ones. However, when the performance of the dynamic policy is compared with the *best* static policy, the latter performs better. This may be due to the fact that a dynamic policy has three parameters that are set heuristically. Further, in a long time range, the distribution of the observed performance of these policies in reference to their real performance becomes large; hence, it is hard in duration-scheduling problems for the dynamic policy to outperform *all* other static policies at *all* times.

Table 6 shows the result obtained for the sample-allocation problems, assuming that the evaluation cost is expensive ($t'_\beta \approx t_\beta$). The first column is the normalized performance averaged over 1,000 simulation runs. The second column is the number of simulation runs where the static policy performs better than the dynamic policy. Here, we set the duration time for the dynamic policy to be $T = 2 \times M$.

Again, the dynamic policy performs better than the static policies on the average, and it performs better than the *best* static policy in most cases. Note that the dynamic sample-allocation policy does not need any heuristic parameters.

The results obtained here extends the results of Fitzpatrick and Grefenstette (1988), who showed that when the duration time is given, a static policy with sample size $N = 1 \sim 2$ performs the best among policies with constant sample sizes. This corresponds to the results in the second row of Table 6. In addition to existing results, our experiments show

Table 5. Comparison of static and dynamic duration-scheduling policies.

policies applied in the simulation	normalized performance value	number of points where $Y_{s(N)} > Y_d$
dynamic (reference)	1.00	–
static ($N = 1$)	1.20	74 / 250
static ($N = 2$)	1.05	110 / 250
static ($N = 4$)	1.06	103 / 250
static ($N = 8$)	1.25	56 / 250
static (best)	0.92	179 / 250
static (mean)	1.14	56 / 250

Table 6. Comparison of static and dynamic sample-allocation policies.

policies applied in the simulation	normalized performance value	number of points where $Y_{s(N)} > Y_d$
dynamic (reference)	1.00	–
static ($N = 1$)	1.33	65 / 1000
static ($N = 2$)	1.25	90 / 1000
static ($N = 4$)	1.27	76 / 1000
static ($N = 8$)	1.45	34 / 1000
static (best)	1.14	198 / 1000
static (mean)	1.32	13 / 1000

that further improvement is possible by applying variable sample sizes. The method is particularly effective when the sampling cost is expensive (and accordingly, the generation cost is negligible).

8. Conclusions

In this paper, we have studied the scheduling of computational resources for genetic algorithms operating in a noisy environment. Assuming that the number of candidates tested in each generation is fixed, we have studied two related scheduling problems. The first problem entails the determination of a suitable duration for a generation, whereas the second requires finding the appropriate number of samples to be drawn from each candidate. Our results show that dynamic scheduling policies perform better than existing static ones.

Although we have used simple test functions in our experiments, the advantage of using dynamic scheduling is universal and is independent of the reproduction mechanism of individual genetic algorithms. (In the extreme case, dynamic policies could improve the performance of random searches.) We are in the process of incorporating our dynamic scheduling policies presented in this paper in a prototype heuristic learning system (Wah, 1992; Ieumwananonthachai, Aizawa, Schwartz, Wah, & Yan, 1992). Future studies will report results on evaluating these dynamic policies for realistic applications.

References

Aizawa, A. N. & Wah, B. W. (1993). Dynamic control of genetic algorithms in a noisy environment. *Proceedings of the International Conference on Genetic Algorithms* (pp. 48–55). San Mateo, CA: Morgan Kaufmann.

Aizawa, A. N. & Wah, B. W. (1994). A sequential sampling procedure for genetic algorithms. *Computers and Mathematics with Applications*, 7:9/10, 77–82.

Booker, L. B., Goldberg, D. E., & Holland, J. H. (1990). Classifier systems and genetic algorithms. In J. Carbonell (Ed.), *Machine learning: Paradigm and methods* (pp. 235–282). Cambridge, MA: MIT Press.

Crow, E. L., Davis, F. A., & Maxfield, M. W. (1960). *Statistics manual*. New York: Dover.

DeJong, K. A. (1975). *Analysis of the behavior of a class of genetic adaptive systems*. Ph.D. dissertation, University of Michigan.

De Jong, K. (1988). Learning with genetic algorithms: An overview. *Machine Learning*, *3*, 121–138.

Ferguson, T. S. (1967). *Mathematical statistics: A decision theoretic approach*. New York: Academic Press.

Fitzpatrick, J. M. & Grefenstette, J. J. (1988). Genetic algorithms in noisy environments. *Machine Learning*, *3*:2/3, 101–120.

Ghosh, B. K. & Sen, P. K. (Eds.). (1991). *Handbook of sequential analysis*. New York: Marcel Dekker.

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

Goldberg, D. E., Deb, K., & Clark, J. H. (1991). Genetic algorithms, noise, and the sizing of populations. *IlliGAL*, 91010.

Goldberg, D. E. & Rudnick, M. (1991). Genetic algorithms and the variance of fitness. *Complex Systems*, *5*, 265–278.

Grefenstette, J. J., Ramsey, C. L., & Schultz, A. C. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, *5*, 355–381.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Cambridge, MA: MIT Press.

Ieumwananonthachai, A., Aizawa, A. N., Schwartz, S. R., Wah, B. W., & Yan, J. C. (1992). Intelligent process mapping through systematic improvement of heuristics. *Journal of Parallel and Distributed Computing*, *15*, 118–142.

Lloyd, E. (1984). *Handbook of applicable mathematics Vol. 6: Statistics*. New York: Wiley & Sons Ltd.

Mühlenbein, H., Schomisch, M., & Born, J. (1991). The parallel genetic algorithm as function optimizer. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the International Conference on Genetic Algorithms* (271–278). San Mateo, CA: Morgan Kaufmann.

Wah, B. W. (1992). Population-based learning: A new method for learning from examples under resource constraints. *IEEE Transactions on Knowledge and Data Engineering*, *4*:5, 454–474.