Machine Learning 3: 101–120, 1988
© 1988 Kluwer Academic Publishers – Manufactured in The Netherlands

Genetic Algorithms in Noisy Environments

J. MICHAEL FITZPATRICK (JMF@VUSE.VANDERBILT.EDU)
JOHN J. GREFENSTETTE[†] (GREFENSTETTE@VUSE.VANDERBILT.EDU)
Computer Science Department, Vanderbilt University, Nashville, Tennessee 37235

(Received: December 23, 1987)
(Revised: May 17, 1988)

Keywords: Learning in noisy environments, genetic algorithms, image registration.

Abstract. Genetic algorithms are adaptive search techniques which have been used to learn high-performance knowledge structures in reactive environments that provide information in the form of payoff. In general, payoff can be viewed as a noisy function of the structure being evaluated, and the learning task can be viewed as an optimization problem in a noisy environment. Previous studies have shown that genetic algorithms can perform effectively in the presence of noise. This work explores in detail the tradeoffs between the amount of effort spent on evaluating each structure and the number of structures evaluated during a given iteration of the genetic algorithm. Theoretical analysis shows that, in some cases, more efficient search results from less accurate evaluations. Further evidence is provided by a case study in which genetic algorithms are used to obtain good registrations of digital images.

1. Introduction

In approaches to machine learning that use genetic algorithms, success depends on the ability to search efficiently for high-performance knowledge structures despite the presense of considerable amounts of noise in the evaluation process. In the approach taken in the systems LS-1 (Smith, 1983) and LS-2 (Schaffer & Grefenstette, 1985), genetic operators are applied to entire production-system programs, based on estimates of each program's overall performance in the task domain. In general, exhaustive evaluations of each candidate program is infeasible. For example, in one series of experiments in which LS-1 learned to play draw poker, each production-system program was evaluated by playing a certain number of hands against a fixed opponent. Given that the hands were generated randomly, this procedure can be viewed as sampling from the performance space of all hands that could be played by the program under evaluation. Consequently, the evaluation process was necessarily noisy.

Likewise, in the classifier-system approach to learning with genetic algorithms (Holland, Holyoak, Nisbett, & Thagard, 1986; Wilson, 1987), genetic

[†]Author's current address: Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC 20375-5000, U.S.A.

operators are applied to individual rules (*classifiers*) based on noisy estimates of each rule's utility (or *strength*). Wilson (1987) has described BOOLE, a system that learned to solve difficult Boolean functions from payoff information. Each iteration through BOOLE's performance cycles produced an updated estimate for the strength of selected classifiers. Genetic operators, invoked at varying intervals between performance cycles, acted on the basis of the current strength assigned to classifiers. Again, each performance cycle can be viewed as a sample from the performance space of the current set of classifiers. This paper addresses the performance of genetic algorithms in the kind of noisy environments that typically confront systems that learn from payoff information.

The issue of efficient search in noisy environments has implications for applications of genetic algorithms beyond machine learning, as well. Although successful pilot studies of genetic algorithms have been completed in a variety of domains, including image processing, combinatorial optimization, gas pipeline control systems, VLSI layout, communication network design, and machine learning (Davis, 1987; Grefenstette, in press), one hindrance to the widespread application of genetic algorithms to large practical problems is that they generally require the evaluation of thousands of candidate solutions. In large, complex search spaces (e.g., high-dimensional, discontinuous spaces with many local optima) genetic algorithms exhibit an impressive improvement over both various forms of random search and local search techniques (De Jong, 1975). Nevertheless, if the evaluation of each candidate solution is computationally expensive, genetic algorithms may not provide a feasible approach. For many problems of practical interest, it is possible to evaluate individual candidate solutions approximately using statistical sampling techniques. In this paper we adopt this general formulation of the problem and investigate the use of genetic algorithms to optimize black-box functions for which approximate function values can be obtained by sampling methods.

In a previous paper (Grefenstette & Fitzpatrick, 1985), we investigated some aspects of the performance of genetic algorithms when candidate solutions are evaluated approximately. In particular, we established that improved performance could result from decreasing the effort applied to accurate function evaluations and increasing the number of iterations of the genetic algorithm, while keeping the size of the current candidate set, or *population*, fixed. This paper considers the effects of varying both the population size and the sampling effort.

The principles of genetic algorithms are presented in the next two sections. Section 4 presents a theoretical analysis of the effects of approximate function evaluations on the search heuristic embodied by genetic algorithms. Section 5 describes experiments that explore the efficiency of genetic algorithms with approximate function evaluations. Section 6 describes an application of these techniques to an image-processing problem: the registration of two digital images. There we define image difference to be the mean of the absolute difference between image intensities at corresponding points in two images. We show that genetic algorithms can be used to search a space of image transformations in order to minimize image differences.

Table 1. A genetic algorithm.

```
procedure genetic algorithm
begin
  $t = 0$;
  initialize $P(t)$;
  evaluate $P(t)$;
  while (not termination condition) do
  begin
    $t = t + 1$;
    select $P(t)$ from $P(t-1)$;
    recombine $P(t)$;
    evaluate $P(t)$;
  end
end.
```

2. A review of genetic algorithms

Genetic algorithms are adaptive generate-and-test procedures derived from principles of natural population genetics. This section presents a high-level description of one formulation of genetic algorithms. Detailed descriptions are given by Holland (1975), Grefenstette (1986), and Goldberg (1988). A skeleton of a simple genetic algorithm is shown in Table 1.

During iteration t, the genetic algorithm maintains a *population* $P(t)$ of *structures* $\{x_1^t, x_2^t, \ldots, x_N^t\}$ chosen from the domain of the objective function f. The initial population $P(0)$ is usually chosen at random. The population size N remains fixed for the duration of the search. Each structure x_i^t is *evaluated* by computing $f(x_i^t)$. Often, the term *trial* is used for each such evaluation. This provides a measure of *fitness* of the evaluated structure for the given problem. When each structure in the population has been evaluated, a new population of structures is formed in two steps. First, structures in the current population are selected to reproduce on the basis of their relative fitness. That is, the *selection* algorithm chooses structures for replication by a stochastic procedure that ensures that the expected number of offspring associated with a given structure x_i^t is $f(x_i^t)/\mu(P,t)$, where $f(x_i^t)$ is the observed performance of x_i^t and $\mu(P,t)$ is the average performance of all structures in the population. That is, structures that perform well may be chosen several times for replication and structures that perform poorly may not be chosen at all. In the absence of any other mechanisms, this selective pressure would cause the best-performing structures in the initial population to occupy a larger and larger proportion of the population over time.

Next the selected structures are recombined using idealized *genetic operators* to form a new set of structures for evaluation. One of the most important genetic operators is *crossover*, which combines the features of two *parent* structures to form two similar *offspring*. Crossover operates by swapping corresponding segments of a string or list representation of the parents. For example, if the parents are represented by five-item lists, say $x_i^t = (a_i\ b_i\ c_i\ d_i\ e_i)$

and $x_j^t = (a_j\ b_j\ c_j\ d_j\ e_j)$, then crossing the segments between the second and the fifth components would produce the offspring $(a_i\ b_i\ c_j\ d_j\ e_i)$ and $(a_j\ b_j\ c_i\ d_i\ e_j)$. Specific decisions define a range of alternative implementations, including whether both resulting structures are to be entered into the population, whether the parents are to be retained, and which other structures, if any, are to be purged.

In generating new structures for testing, the crossover operator draws only on the information present in the structures of the current population. If specific information is missing, due to storage limitations or loss incurred during the selection process of a previous iteration, then crossover is unable to produce new structures that contain it. A *mutation* operator that arbitrarily alters one or more components of a selected structure provides the means for introducing new information into the population. However, in contrast to the early computational models of evolutionary processes (Fogel, Owens, & Walsh, 1966), mutation functions solely as a background operator within a genetic algorithm (i.e., its probability of application is kept very low). Its presence ensures that all points in the search space can be reached.

3. Implicit parallelism of genetic algorithms

The power of the adaptive search strategies described above does not lie in the testing of individual structures; rather, it resides in the efficient exploitation of the wealth of information that the testing of structures provides with regard to the interactions among the components of these structures. Specific configurations of component values observed to contribute to good performance (e.g., values of a specific pair of list items in the example above) are preserved and propagated through the structures in the population in a highly parallel fashion. The development of successful small configurations, in turn, forms the basis for subsequent exploitation of larger and larger such configurations. Intuitively, one can view these structural configurations as the regularities in the space that emerge as individual structures are generated and tested. Once encountered, they serve as *building blocks* in the generation of new structures.

More specifically, let H be a *hyperplane* in the search space. Suppose that the elements of the five-item lists above are allowed to take only the values 0 or 1. Then the search space is a five-dimensional binary space. For example, the hyperplane denoted by $H = 0\#\#\#1$ consists of all structures that have a 0 in position one and a 1 in position five. Let $M(H,\ t)$ denote the number of structures in $P(t)$ that are also members of H. Holland (1975) has shown that the result of selection alone (in the absence of genetic operators) is a new distribution of hyperplanes that obeys the formula

$$M(H,\ t+1) = \frac{\mu(H,\ t)}{\mu(P,\ t)} M(H,\ t), \tag{1}$$

where $\mu(H,\ t)$ is the average fitness of the structures that are in both $P(t)$ and in H, and $\mu(P,\ t)$ is the average fitness of all structures in $P(t)$. Equation (1) says that the number of trials allocated to an above-average hyperplane H grows exponentially over time. The crucial observation is that (1) applies to each hyperplane H that is represented in population P. The following proposition shows that a large number of distinct hyperplanes are allocated trials according to (1).

Proposition (Holland, 1980). *Consider a population P of N random binary structures of length L. For most practical values of N and L, at least N^3 hyperplanes are allocated trials according to (1).*

Proof. See Goldberg (1985). Appendix A presents an alternative proof.

For example, if $N = 128$ and $L = 64$ we can expect that at least ten million distinct hyperplanes are each represented by at least eight structures in P. The number of structures representing each of these ten million hyperplanes will vary according to (1).

That is, genetic algorithms actually search the space of all feature combinations, quickly identifying and exploiting combinations associated with high performance. The ability to perform such a search by evaluating completely specified candidate solutions is called the *implicit parallelism* of genetic algorithms.

Of course, achieving the above exploration rate with respect to a given hyperplane H is dependent on the specific configuration of component values that defines the hyperplane passing unchanged from parent to offspring. Hence, it is necessary to examine the disruptive effects of the genetic operators. The crossover operator will disrupt a given structural configuration if the selected crossover point falls between two or more of the configuration's component values. The probability of this occurring is directly proportional to the length of the smallest sequence of components containing the configuration, or the *defining segment* of the corresponding hyperplane. For example, the probability of disruption by crossover associated with the hyperplane $H = \#\# \ldots \#1\#01\# \ldots \#\#$ is no more than $3/(L-1)$, where L is the length of the structure representation. Thus, crossover tends to preserve the allocation rate in (1) with respect to hyperplanes whose defining segments are small relative to L, and tends to be disruptive with respect to hyperplanes having large defining segments. However, as structures belonging to specific, high-performance hyperplanes with small defining segments begin to dominate the population over time, the remaining population structures become more similar, and hence crossover has a less disruptive effect on the rate at which hyperplanes are allocated trials. The mutation operator has a negligible effect on the allocation rates, given its background role in the search.

Further theoretical properties of genetic algorithms have been extensively analyzed (De Jong, 1975; Holland, 1975; Goldberg, 1988). It is clear that genetic algorithms make much more extensive use of the information provided by the evaluation of candidates than most other heuristic search methods. For example, a hill-climbing algorithm tests several structures and keeps the most promising one. In the process, hill climbing discards a vast amount of information concerning the combinations of features present in the unsuccessful structures. In genetic algorithms, on the other hand, combinations of features in unsuccessful structures may still be passed along to other more successful structures. To summarize, the power of a genetic algorithm derives from its ability to exploit, in a near-optimal fashion (Holland, 1975), information about the utility of a large number of structural configurations without the computational burden of explicit calculation and storage. This leads to a focused exploration of the search space wherein attention is concentrated in regions that contain structures of above average utility. Nonetheless, the population is widely distributed over the space, insulating the search from susceptibility to stagnation at a local optimum.

4. Genetic algorithms with approximate evaluations

In this section we analyze the impact of approximate evaluations on the heuristic embodied in genetic algorithms. This topic is motivated in part by the desire to apply genetic algorithms to problems whose candidate solutions can be evaluated by statistical methods. This approach is also motivated by the work of De Jong (1975), who included a noisy function as part of his experimental study of genetic algorithms, but did not specifically study the implications for using approximate evaluations on their efficiency. Our main question is the following: given a fixed amount of computation time, is it better to devote substantial effort to getting highly accurate evaluations or to obtain quick, rough evaluations and allow the genetic algorithm to consider many more candidate solutions per iteration? We assume that the evaluation of each structure by the genetic algorithm involves statistical sampling and the effort required for each trial is proportional to the number of samples taken.

Statistical sampling techniques are often used in the evaluation of integrals of complicated integrands over large domains. Such integrals appear in many applications of physics and engineering (James, 1980; Lautrup, 1985). Throughout our discussions it will be convenient to treat the function, $f(x)$, to be optimized as the mean of some random variable $R(x)$. In terms of the evaluation of an integral by statistical sampling, $f(x)$ is the mean of the integrand's value over the domain and $R(x)$ is simply the set of values of the integrand over the domain. The approximation of $f(x)$ by the statistical technique proceeds by selecting n random samples from $R(x)$. The mean of the sample serves as the approximation and, to the extent that the samples are random, the sample mean is guaranteed by the law of large numbers to converge to $f(x)$ with increasing n. Once $f(x)$ is approximated, the desired value of the integral can be approximated by multiplying the approximation of $f(x)$ by the volume of the domain. There are many approaches to improving the convergence of the sample mean and the confidence in the means for a fixed n (Lautrup, 1985), but we will not investigate these approaches. Here we will be concerned only with the sample mean and an estimate of our confidence in that mean.

The main idea is to use as an evaluation function in the genetic optimization of $f(x)$, not $f(x)$ itself, but an estimate $e(x)$ obtained by taking n randomly chosen samples from a random variable $R(x)$ with mean $f(x)$. It seems intuitively clear that $e(x)$ approaches $f(x)$ for large n. Statistical sampling theory tells us that if $R(x)$ has standard deviation $\sigma(x)$ then the standard deviation of the sample mean, $\sigma_s(x)$, is given by

$$\sigma_s(x) = \sigma(x)/\sqrt{n}. \tag{2}$$

It is clear from (2) that reducing the size of $\sigma_s(x)$ can be expensive. Reducing $\sigma_s(x)$ by a factor of two, for example, requires four times as many samples and hence four times as much evaluation time per trial. On the other hand, it is clear that the genetic algorithm will require more trials to reach a fixed level of optimization for $f(x)$ when $\sigma_s(x)$ is larger and that the genetic algorithm will achieve a less satisfactory level of optimization for $f(x)$ for a fixed number of trials when $\sigma_s(x)$ is larger. What is not obvious is which effect is more important, the increase in the number of trials required or the increase in the time required per trial. In Sections 5 and 6 we describe experiments that explore the relative importance of these two effects. Here we consider how the use of approximate evaluation of individual structures affects the estimate of the average performance of the hyperplanes in the search space.

In particular, we derive an expression for the standard deviation of the estimate of the average performance for an arbitrary hyperplane H. This estimate controls the number of offspring allocated to H by the selection algorithm, and hence the focus of the genetic search. For the genetic algorithm to correctly identify high performance areas of the search space, it is important to minimize the error in the estimated average performance of hyperplanes. Let H be an arbitrary hyperplane in the representation space comprising a set of structures

$$H = \{x_1, x_2, \ldots, x_{|H|}\}.$$

The performance of H can be viewed as a random variable with mean μ and variance σ^2. Let E be a random variable defined by randomly choosing r structures from H and computing the average (exact) evaluation of each structure. Then E has mean $\mu_E = \mu$ and variance $\sigma_E^2 = \sigma^2/r$.

We now consider the effects of evaluating the chosen structures by a sampling procedure. Let $R(x_i)$ denote the random variable from which samples are taken in order to compute the estimate $e(x_i)$. Then $R(x_i)$ has mean $\mu(x_i) = f(x_i)$ and variance $\sigma^2(x_i)$. If we estimate $\mu(x_i)$ by taking n samples from $R(x_i)$, the resulting random variable has mean $\mu(x_i)$ and variance

$$\sigma_n^2(x_i) = \frac{\sigma^2(x_i)}{n}. \tag{3}$$

Let S be the random variable that results from averaging the results of taking n samples from each of r randomly chosen x_i's in H. S has mean μ and, as shown in Appendix B, it has variance

$$\sigma_S^2 = \frac{1}{r}\sigma^2 + \frac{1}{rn}\langle\sigma^2(x_i)\rangle_H, \tag{4}$$

where $\langle\ldots\rangle_H$ denotes the mean over all structures in H. Note that

$$\lim_{n\to\infty} \sigma_S^2 = \sigma_E^2.$$

That is, as the number of samples per trial gets large, the difference between approximate evaluation and exact evaluation diminishes, as expected. This analysis suggests a way to improve the estimation of hyperplane performance. Since the total number of samples taken during the estimation of H is rn, (4) suggests that the estimate of μ may be improved at no additional sampling cost by increasing r and decreasing n so as to keep the product rn constant. Since the value of r is proportional to the population size N, it is reasonable to investigate the effect of increasing the population size while decreasing the number of samples per trial.

Before we present our experimental study of this effect, it is instructive to consider the effect of decreasing the number, n, of samples without varying the population size. From (4), we get

$$\frac{\partial \sigma_S^2}{\partial n} = -\frac{1}{rn^2}\langle \sigma^2(x_i)\rangle_H,$$

and from (3) it follows that

$$\frac{\partial \langle \sigma_n^2(x_i)\rangle_H}{\partial n} = -\frac{1}{n^2}\langle \sigma^2(x_i)\rangle_H,$$

giving

$$\frac{\partial \sigma_S^2}{\partial n} = \frac{1}{r}\frac{\partial \langle \sigma_n^2(x_i)\rangle_H}{\partial n}. \tag{5}$$

In words, suppose we estimate the performance of hyperplane H by averaging the observed evaluations of r structures randomly selected from H. As the number n of samples per evaluation increases, the accuracy of the estimated performance of the hyperplane H increases at a rate equal to only $1/r$ times the rate of increase of the average accuracy of the evaluation of individual structures. Conversely, decreasing n results in a smaller percentage loss of accuracy in the estimation of the hyperplane performance than in the evaluation of individual structures. Since the quality of the search performed by genetic algorithms depends on the quality of its estimates of the performance of hyperplanes, rather than the evaluation of particular individual structures, (5) suggests that genetic algorithms can be expected to perform well for problems requiring partial evaluation of candidate solutions.

There are interesting questions concerning the optimal balance between the effort spent on evaluating individual structures and the effort spent on other aspects of the genetic algorithm. Equation (5) suggests that computation time could be saved, with relatively little negative effect on the genetic algorithm, by reducing the number of samples taken per trial. Assuming a fixed amount of time is available, we can utilize the time saved by increasing the population size, by increasing the number of generations, or by doing both. In a previous paper (Grefenstette & Fitzpatrick, 1985), we investigated one aspect of this tradeoff and established that, assuming negligible overhead for the operation of the genetic algorithm, performance can be improved by increasing the number of generations alone. In this paper, we investigate the general case, varying both population size and number of generations and taking the overhead into account.

5. Tradeoffs between population size and sample size

This section empirically explores the effect suggested by analysis in the last section, namely, that the estimation of hyperplanes can be improved by increasing the population size while decreasing the number of samples taken per trial. A genetic algorithm was applied to minimize a test function, described below, on which statistical sampling was performed to evaluate structures in the search space. The primary goal was to study the relationship between population size and sample size when the overall computation time was fixed.

The initial experiments used a genetic algorithm to minimize the function $f(x) = \sum_{j=1}^{30} j a_j^4$, where $x = \langle a_1, \ldots, a_{30} \rangle$, and $-1.28 \le a_j < 1.28$, for $j = 1, 2, \ldots, 30$. De Jong (1975) used $f(x)$ in his early study of the behavior of genetic algorithms, in which he investigated the effects of adding Gaussian noise to the function. In this study, we estimate $f(x)$ by sampling; in particular, we treat $f(x)$ as the mean of a normal distribution with $\sigma = 2.0$. Thus, an important difference between this study and De Jong's is that the earlier one assumed a fixed amount of noise, whereas the amount of noise in these experiments depends on the number of samples taken.

5.1 Overhead considerations and experimental design

Our primary concern is with improving the performance of genetic algorithms when time is limited. With this purpose in mind, the following experiments ignore the cost of the extra memory required to accommodate increased population sizes, though we do restrict our attention to population sizes that can be easily accommodated on currently available microcomputers.

In order to accurately assess the time-related tradeoffs between taking more samples per trial and evaluating more structures per generation, the overhead resulting from the genetic algorithm must be taken into account. For a given representation of structures, each of the phases *initialize*, *select*, and *recombine* shown in Table 1 takes a constant amount of time per evaluated structure.[1] Let α be total computational cost that can be attributed to the genetic algorithm per trial. Let β be the cost per sample taken during the evaluation of each structure. Then the total time required for the genetic algorithm, neglecting the initialization phase, is

$$T = (\alpha + \beta n) G N,$$

where G is the total number of generations, N is the population size, and n is the number of samples per trial. Consider the effects of increasing N while keeping T and Nn constant. Clearly, the number of generations that can be achieved in a fixed amount of time decreases as we increase the population size. Furthermore, the rate at which G decreases depends on the ratio α/β. That is, we expect to see the following tradeoff: increasing the population size while keeping Nn constant should improve the quality of the information

[1]Baker (1987) presents an algorithm for selecting the offspring of successive generations in linear time.

processed by the genetic algorithm, as shown in (4). At the same time, the genetic algorithm will have fewer iterations G in which to use the improved information.

A series of experiments was performed in order to assess the tradeoff between the population size and sample size for various values of α/β. In each experiment, the total computation time T and the product Nn were held fixed, while N varied between 25 and 2000 and n varied between 80 and 1. The value for T was chosen so that the number of generations G was 200 for $N = 25$.

5.2 Performance metric

Previous studies (De Jong, 1975; Grefenstette, 1986) have suggested ways to measure both the quality of the final result and the efficiency of the intermediate behavior of genetic algorithms. In comparing the various runs of a genetic algorithm using approximate evaluation techniques, care must be taken in defining a useful performance statistic. It would be misleading to simply record the best evaluation of all structures examined during the search, since the accuracy of the evaluation varies with the number of samples. For example, if the number of samples is very small, there may be little relation between the observed performance on any individual structure and that structure's true performance when exactly evaluated. If we consider that the entire final population represents the knowledge obtained by the genetic algorithm, it seems natural to measure the performance of a genetic algorithm by the best exact evaluation of all structures in the final population. However, this metric has the disadvantage that it requires more computational effort to exactly evaluate a large population than a small one. Thus the total computation time devoted to running and evaluating a genetic algorithm with large N is greater than if N is small. In order to determine whether this difference could be eliminated, we collected another performance statistic in the experiments, consisting of the best exact evaluation of the top 50 structures in the final population, selected on the basis of the observed performance values. This technique appears to provide a practical way to measure the performance of a genetic algorithm with a large population.

5.3 Experimental results

For each experiment at least ten runs of the genetic algorithm were performed, using different random number seeds, on the test function described above for each combination of N and n. The average of the performance metric for these runs is shown in Tables 2 through 4. Note that since the function is being minimized, a smaller performance value indicates a more successful search strategy.

In the first experiment, the ratio α/β was set to zero, corresponding to a situation in which the cost of sampling far surpasses the cost per structure associated with the genetic algorithm. This situation arises in practice when the evaluation of a structure requires running a complex simulation (Grefenstette, 1986) or a production-system interpreter (Smith, 1983; Grefenstette, 1987). Table 2 shows the results.

Table 2. Results of the first experiment: Average performance of best structures on the test function when $\alpha/\beta = 0$.

NUMBER OF SAMPLES	SIZE OF POPULATION	NUMBER OF GENERATIONS	AVERAGE PERFORMANCE
1	2000	200	1.38
2	1000	200	1.30
5	400	200	1.48
10	200	200	1.59
20	100	200	1.79
40	50	200	2.55
80	25	200	4.56

Since the overhead for the genetic algorithm is insignificant in this case, the number of generations is identical (200) for all runs in this experiment. That is, the genetic algorithm has an equal number of iterations in which to operate. However, the accuracy of the estimated hyperplane improves as N increases. In this case, the results are consistent with our theoretical analysis in Section 2. (The difference in performance between using a sample size of one and a sample size of two is not statistically significant.) The best results are achieved with smaller sample sizes and larger population sizes. As n increases and N decreases, performance declines as expected, with a dramatic drop at $n > 20$. This is because, as n increases beyond 20, the improvement in the accuracy of each evaluation does not compensate for the reduction in the number of representatives for each hyperplane in the smaller population. We now consider cases where the genetic algorithm overhead plays a more significant role.

In the second experiment, the ratio α/β is assumed to be three. This corresponds to a moderate cost per sample, as might be expected when computing a moderately complex numerical function. This is also approximately the ratio associated with the image-processing problem described in the next section. The results for the test function are shown in Table 3. In this experiment, there is no statistically significant difference in performance between the runs at 5, 10, and 20 samples. However, the difference between the results with 5 samples and the result with fewer samples is statistically significant at the 0.05 level. This experiment shows that, when genetic algorithm overhead is taken into account, a secondary effect comes into play when N is large. Namely, the number of generations possible within the available computation time declines, leading to a corresponding decline in performance. This experiment shows that it is necessary to balance the time spent per generation and the number of generations performed when there is significant overhead for the genetic algorithm.

In the third experiment, the ratio α/β is assumed to be 32, which is in fact the ratio of overhead to sampling effort in the test function. The results for this experiment are shown in Table 4. As expected from the discussion of the previous experiment, these data reveal a severe tradeoff between time spent per generation and total number of generations. In this experiment, the opti-

Table 3. Results of the second experiment: Average performance of best structures on the test function when $\alpha/\beta = 3$.

NUMBER OF SAMPLES	SIZE OF POPULATION	NUMBER OF GENERATIONS	AVERAGE PERFORMANCE
1	2000	52	5.27
2	1000	83	3.19
5	400	130	2.10
10	200	160	2.09
20	100	180	1.96
40	50	193	2.63
80	25	200	4.71

mal performance occurs when between 20 and 40 samples are taken per trial. Runs with fewer samples per trial had larger population sizes and therefore far fewer generations in the allotted time. In these cases, the relatively high additional costs incurred by the genetic algorithm exacerbated the problem of allowing it enough generations to perform a thorough search. It is interesting to note that performance also declines if the population is too small. When the population size falls below 50, the increase in the number of generations does not compensate for the decrease in both the number of hyperplanes processed per generation (Proposition 1) and the accuracy of the performance estimates of the hyperplanes (Equation 5).

In summary, these preliminary experiments support the analysis of trading additional samples per trial for additional structures per population. In general, increasing the population size while decreasing the samples per trial appears to improve the performance of genetic algorithms. However, this effect is moderated by the requirement that the algorithm perform sufficient numbers of iterations to adequately explore the search space. This latter requirement places a limit on the population size when the genetic algorithm overhead is relatively high in comparison to the sampling cost.

6. Tests on an image registration problem

This section describes experiments using approximate evaluation on a realistic task – *image registration*. The general problem of image registration is important in such diverse fields as aerial photography (Svedlow, McGillem, & Anuta, 1978; Merchant, 1981) and medical imaging (Venot & Leclerc, 1984; Fitzpatrick, Pickens, Grefenstette, Price, & James, 1987). General introductions to the field of image registration and extensive bibliographies may be found in Hall (1979) and Goshtasby (1983). An image comparison technique based on random sampling, different from the method used here, is described by Barnea and Silverman (1972).

The function to be minimized in image registration measures the difference between two images of a scene, in our case medical x-ray images, that have

Table 4. Results of the third experiment: Average performance of best structures on the test function when $\alpha/\beta = 32$.

NUMBER OF SAMPLES	SIZE OF POPULATION	NUMBER OF GENERATIONS	AVERAGE PERFORMANCE
1	2000	8	41.32
2	1000	16	27.33
5	400	38	10.47
10	200	67	5.73
20	100	108	3.62
40	50	156	3.20
80	25	200	4.71

been acquired at different times. The images differ because of motion that has taken place between the two acquisition times, because of the injection of dye into the arteries, and because of noise in the image acquisition process. The registration of such images is necessary for the successful use of *digital subtraction angiography*, in which an image of an artery's interior is produced by subtracting a pre-injection image from a post-injection image. Fitzpatrick et al. (1987) describe the details of the process and the registration technique. The misregistration of two images results in motion artifacts such as those in Figure 1. By performing a geometrical transformation that warps one image relative to the other, it is possible to improve the registration of the images so that the difference due to motion is reduced. For example, Figure 2 shows an improved registration resulting from a geometrical transformation prior to subtraction. We have investigated the use of a genetic algorithm to search a parameterized space of image transformations in order to minimize the image difference. The class of transformations we consider includes elastic motion, rotation, and translation.

The method selects two images and designates one of them as the *mask* image. A transformed version of that image is to be compared to a second image – the *target* image – within a square subimage – the *region of interest.* (The target image typically includes dye injected into the arteries.) The space of transformations is parameterized by four vectors – d_1, d_2, d_3, and d_4 – that specify the motion of the mask image at the four corners of the region of interest. The motion of intermediate points is determined by means of bilinear interpolation from the corner points. (More complicated warpings can be described with additional vectors.)

The images are represented digitally as square arrays of *pixels* representing an approximate map of image intensity. The image difference is defined to be the mean absolute difference between the pixels at corresponding positions in the transformed and target images. The exact mean can be determined by measuring the absolute difference at each pixel position; an estimate of the mean may be obtained by sampling randomly from the population of absolute pixel differences. The effort required to estimate the mean is approximately

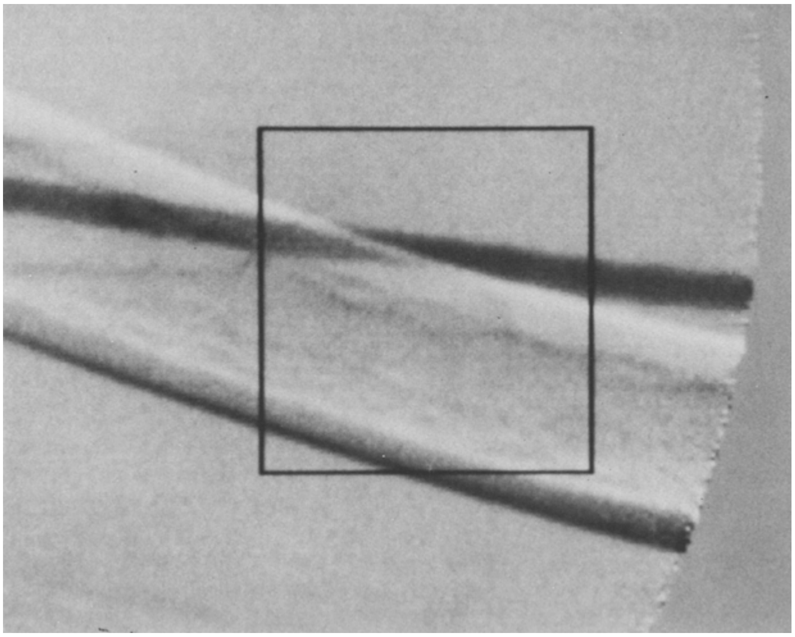

Figure 1. A difference image of the region surrounding the humerus. An artery is made visible (dark horizontal band) by injecting a dye that is relatively opaque to x-rays. The image was obtained by subtracting a pre-injection image, the *mask*, from a post-injection image. Motion artifacts resulting from misregistration of bone (dark and light diagonal bands) interfere with the image of the artery. The square outlines a 100 by 100 pixel region of interest. The mask image must be transformed within this region to reduce the severity of the artifacts.

proportional to the number of samples taken, so this problem meets the conditions for statistical evaluation of candidate solutions. The region of interest is typically 100 by 100 pixels, giving a sample space of size 10,000 values from which to estimate the exact mean. The parameters for the transformation comprise the x and y components of the four vectors. The magnitude of each component is limited to less than one-fourth the width of the region of interest, to avoid the possibility of folding (Fitzpatrick & Leuze, 1987). The range for each of these eight components is digitized to eight-bit accuracy.

Table 5 shows the results from a series of experiments in which genetic algorithms were applied to eight-image registration problems, generated by selecting four regions from each of two pairs of x-ray images, one of the elbow and one of the humerus. The unregistered difference image for the eighth region, a segment of the humerus, is shown in Figure 1. As in the earlier experiments, the population size was varied in order to keep the sampling time per generation fixed across all runs. The total time per run was fixed at five minutes (on

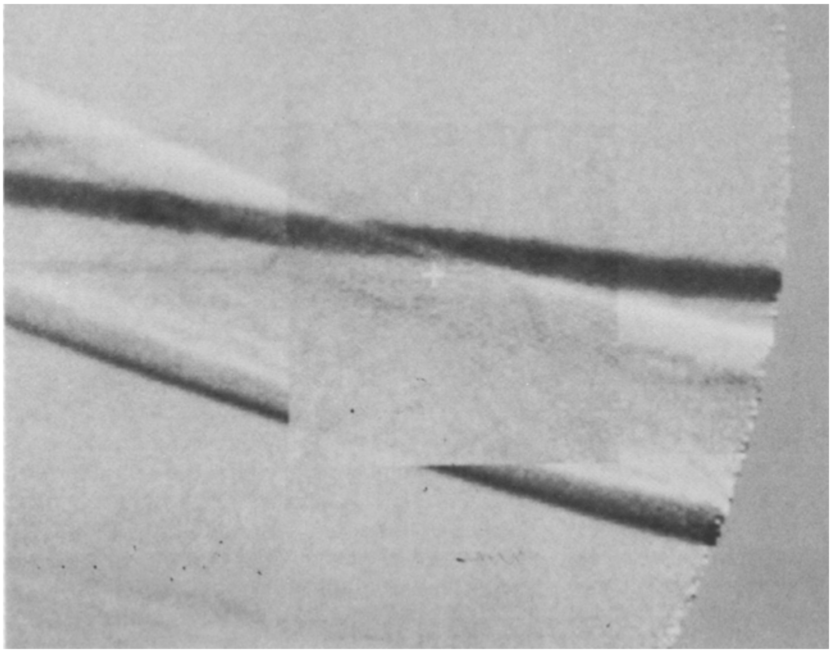

Figure 2. A difference image of the same region as Figure 1. Here the region of interest, outlined in Figure 1, shows reduced motion artifacts. The image of the artery is much improved where it crosses the bone edge (above and to the left of the + sign). The misregistration in the lower left corner of the region of interest is almost completely removed. The improvement is accomplished by transforming the mask image within this region before subtracting.

a Sun 3/260 workstation with floating point accelerator). The performance of each genetic algorithm was measured according to the technique described previously: the 50 structures with the best observed performance in the final population were subjected to an exact computation of the mean pixel difference between the target image and the corresponding transformed image. The results are normalized so that random search gets a score of 1.00 on each image and lower scores indicate improved search performance. The registration shown in Figure 2 was found during one of the runs of the genetic algorithm reported in column I_8 in Table 5, using ten samples per trial. Table 5 shows the average results from 50 runs of the genetic algorithm for each sample size per evaluation. The results in Table 5 are consistent with the results shown in Table 3 for the corresponding α/β ratio in the test function.

There is statistically significant improvement for all images when the sample size is increased from one to two, and again when it is increased from two to five. The best results are obtained for five to ten samples per evaluation;

Table 5. Performance of best structures on eight image registration problems.

SAMPLES	POPUL.	GENS.	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
1	2000	103	.84	.88	.75	.86	.81	.81	.87	.79
2	1000	163	.53	.69	.47	.55	.71	.69	.77	.58
5	400	251	.28	.45	.39	.33	.66	.61	.56	.50
10	200	307	.27	.35	.39	.32	.66	.61	.48	.50
20	100	343	.32	.33	.40	.34	.68	.62	.49	.51
40	50	366	.34	.36	.45	.36	.70	.65	.51	.54
80	25	378	.43	.47	.52	.43	.73	.67	.55	.59

increasing the samples beyond ten (and decreasing the population size accordingly) decreases performance significantly. The slight difference between the results in Table 3 and Table 5 can be explained on the basis of our previous analysis. First, the α/β ratio for the registration problem is actually a little less than three. Second, the total time per run in the registration studies allowed nearly twice as many generations as in the test case. Both of the factors favor taking slightly fewer samples than in the second experiment. However, the overall similarity of the results between the test studies and the image-registration study supports our analysis of the behavior of genetic algorithms with approximate evaluations of candidate solutions.

7. Conclusions

Genetic algorithms search by allocating effort to regions of the search space based on an estimate of the relative performance of competing regions. One benefit of this approach is that the individual knowledge structures representing the competing regions of the space need not be evaluated precisely. This observation lets one apply genetic algorithms to problems in which the environment provides noisy or approximate payoff information, or in which the evaluation of knowledge structures can only be performed through statistical techniques. Our analysis suggests that in some cases the overall efficiency of genetic algorithms may be improved by reducing the time spent on individual evaluations and by increasing the population size. This analysis has been supported by a case study in the image-processing domain.

This work has important implications for approaches to machine learning that search for high-performance knowledge structures. Even in limited domains (Smith, 1983), it is typically impossible to evaluate precisely the performance of a given program or rule. In a general-purpose learning system (Holland et al., 1986), one expects perpetual novelty to be a characteristic feature of many learning environments. In these cases, traditional search techniques such as hill climbing are likely to be misled by noise in the available environmental feedback. This work suggests that genetic algorithms may be the search technique of choice for machine learning systems in complex environments.

Acknowledgements

This research was supported in part by grants from the National Institutes of Health (HL34703), the National Science Foundation (ECS-8608588), and the Whitaker Foundation. We acknowledge the assistance of Dirk Van Gucht in the early stages of this project. We also wish to thank the editors for their help in the final preparation of this paper.

References

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 14–21). Cambridge, MA: Lawrence Erlbaum.

Barnea, D. I., & Silverman, H. F. (1972). A class of algorithms for fast digital image registration. *IEEE Transactions on Computers, 212*, 179–186.

Davis, L. (1987) (Ed.). *Genetic algorithms and simulated annealing.* London: Pitman Press.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

Fitzpatrick, J. M., & Leuze, M. R. (1987). A class of injective two dimensional transformations. *Computer Vision, Graphics, and Image Process, 39*, 369–382.

Fitzpatrick, J. M., Pickens, D. R., Grefenstette, J. J., Price, R. R., & James, A. E. (1987). A technique for automatic motion correction in DSA. *Optical Engineering, 26*, 1085–1093.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution.* New York: John Wiley.

Goldberg, D. E. (1985). *Optimal initial population size for binary-coded genetic algorithms* (TCGA Report No. 85001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

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

Goshtasby, A. (1983). *A symbolically-assisted approach to digital image registration with application in computer vision.* Doctoral dissertation, Department of Computer Science, Michigan State University, East Lansing.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 16*, 122–128.

Grefenstette, J. J. (1987). Multilevel credit assignment in a genetic learning system. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 202–209). Cambridge, MA: Lawrence Erlbaum.

Grefenstette, J. J. (in press). Genetic algorithms and their applications. In A. Kent & J. G. Williams (Eds.), *Encyclopedia of Computer Science and Technology* (Vol. 21, Supplement 6). New York: Marcel Dekker, Inc.

Grefenstette, J. J., & Fitzpatrick, J. M. (1985). Genetic search with approximate function evaluations. *Proceedings of the First International Conference on Genetic Algorithms and Their Applications* (pp. 112–120). Pittsburgh, PA: Lawrence Erlbaum.

Hall, E. L. (1979). *Computer image processing and recognition.* New York: Academic Press.

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

Holland, J. H. (1980). Adaptive algorithms for discovering and using general patterns in growing knowledge-bases. *International Journal of Policy Analysis and Information Systems*, *4*, 217–240.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of inference, learning, and discovery.* Cambridge, MA: MIT Press.

James, F. (1980). Monte Carlo theory and practice. *Reports on Progress in Physics*, *43*, 73.

Lautrup, B. (1985). Monte Carlo methods in theoretical high-energy physics. *Communications of the ACM*, *28*, 358–373.

Merchant, J. (1981). Exact area registration of different views of a common object scene. *Optical Engineering*, *20*, 424–436.

Schaffer J. D., & Grefenstette, J. J. (1985). Multi-objective learning via genetic algorithms. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 593–595). Los Angeles, CA: Morgan Kaufmann.

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 422–425). Karlsruhe, West Germany: Morgan Kaufmann.

Svedlow, M., McGillem, C. D., & Anuta, P. F. (1978). Image registration: Similarity measure and preprocessing method comparisons. *IEEE Transactions on Aerospace and Electronic Systems*, *14*, 141–150.

Venot, A., & Leclerc, V. (1984). *IEEE Transactions on Medical Imaging*, *3*, 179–186.

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning*, *2*, 199–228.

Appendix A. Implicit parallelism in genetic algorithms

The following result is one of the folk theorems of genetic algorithms, and was first announced by Holland (1980). The proof included here is somewhat shorter than Goldberg's (1985), but focuses only on the effects of selection and ignores the effects of genetic operators.

Proposition. *Consider a population P of N random binary structures of length L. For most practical values of N and L, at least N^3 hyperplanes are allocated trials according to (1).*

Proof. First note that (1) is a reasonable heuristic for a given hyperplane H only to the extent that $\mu(H,t)$ gives a reasonable measure of the quality of H. We therefore restrict our enumeration to those hyperplanes that have some minimum number, say r, of representatives in P. Let $k = \log(N/r)$. Then for any choice of k positions, there are 2^k distinct hyperplanes defined at those k positions, each of which can be expected to be represented by r structures in P. Therefore, the number of distinct hyperplanes with r representatives in P is at least

$$M_r = 2^k \times \binom{L}{k}. \tag{6}$$

For most problems of practical interest, $L \geq 64$ and $2^6 \leq N \leq 2^{20}$, and it is reasonable to require that $r \geq 8$. If $r = 8$, then $3 \leq k \leq 17$. By inspection of the value of (6) over this range, we find that $M_r \geq N^3$. ■

Appendix B. Derivation of Equation (5)

Here, we derive the variance σ_S^2 for S, the random variable that results from averaging n samples from each of r x_i's randomly chosen from H. Let $p_j(x_i)$ denote the jth sample taken from the random variable $R(x_i)$ in the estimation of $f(x_i)$, for $j = 1,\ldots,n$. Then the mean performance of x_i is

$$\mu(x_i) \equiv \langle p_j(x_i)\rangle_R = f(x_i),$$

and the mean performance of hyperplane H is

$$\mu \equiv \langle \mu(x_i)\rangle_H,$$

where $\langle\ldots\rangle_H$ denotes the mean averaged over all possible sets of r members of hyperplane H and $\langle\ldots\rangle_R$ denotes the mean averaged over all possible sets of n samples taken from $R(x_i)$. It follows that

$$\mu(x_i) = \mu + \eta(x_i),$$

where $\langle \eta(x_i)\rangle_H = 0$ and

$$p_j(x_i) = \mu(x_i) + \eta_j(x_i),$$

where $\langle \eta_j(x_i)\rangle_R = 0$. Note that the variance associated with H is

$$\sigma^2 = \langle \eta(x_i)^2\rangle_H$$

and that the variance associated with each x_i is

$$\sigma^2(x_i) = \langle \eta_j(x_i)^2\rangle_R.$$

It follows that

$$\begin{aligned}
\sigma_S^2 &= \langle\, \langle |\frac{1}{r}\sum_{i=1}^{r}[\frac{1}{n}\sum_{j=1}^{n}(p_j(x_i)) - \mu]|^2\rangle_R\rangle_H \\
&= \frac{1}{r^2n^2}\langle\, \langle |\sum_{i=1}^{r}[\sum_{j=1}^{n}(\mu(x_i) + \eta_j(x_i)) - n\mu]|^2\rangle_R\rangle_H \\
&= \frac{1}{r^2n^2}\langle\, \langle |\sum_{i=1}^{r}[n\mu(x_i) - n\mu + \sum_{j=1}^{n}\eta_j(x_i)]|^2\rangle_R\rangle_H \\
&= \frac{1}{r^2n^2}\langle\, \langle |\sum_{i=1}^{r}[n\eta(x_i) + \sum_{j=1}^{n}\eta_j(x_i)]|^2\rangle_R\rangle_H \\
&= \frac{1}{r^2n^2}\langle\, \langle \sum_{i=1}^{r}[n\eta(x_i) + \sum_{j=1}^{n}\eta_j(x_i)]\sum_{i'=1}^{r}[n\eta(x_{i'}) + \sum_{j'=1}^{n}\eta_{j'}(x_{i'})]\rangle_R\rangle_H) \\
&= \frac{1}{r^2n^2}(\sum_{i=1}^{r}\sum_{i'=1}^{r}n^2\langle\langle\eta(x_i)\eta(x_{i'})\rangle_R\rangle_H + 2\sum_{i=1}^{r}\sum_{i'=1}^{r}\sum_{j'=1}^{n}n\langle\langle\eta(x_i)\eta_{j'}(x_{i'})\rangle_R\rangle_H \\
&\quad + \sum_{i=1}^{r}\sum_{i'=1}^{r}\sum_{j=1}^{n}\sum_{j'=1}^{n}\langle\, \langle\eta_j(x_i)\eta_{j'}(x_{i'})\rangle_R\rangle_H).
\end{aligned}$$

If we assume that $\eta(x_i)$ and $\eta(x_{i'})$ are uncorrelated, we may simplify the first term,

$$\begin{aligned}
\langle\, \langle\eta(x_i)\eta(x_{i'})\rangle_R\rangle_H &= \langle\eta(x_i)\eta(x_{i'})\rangle_H \\
&= \delta_{ii'}\sigma^2.
\end{aligned}$$

If we assume further that the $\eta_j(x_i)$ and the $\eta_{j'}(x_{i'})$ are uncorrelated, we may simplify the third term,

$$\begin{aligned}
\langle\, \langle\eta_j(x_i)\eta_{j'}(x_{i'})\rangle_R\rangle_H &= \delta_{jj'}\delta_{ii'}\langle\, \langle\eta_j(x_i)\eta_{j'}(x_{i'})\rangle_R\rangle_H \\
&= \delta_{jj'}\delta_{ii'}\langle\sigma^2(x_i)\rangle_H.
\end{aligned}$$

Because $\langle\eta_{j'}(x_{i'})\rangle_H = 0$, we may simplify the second term,

$$\begin{aligned}
\langle\, \langle\eta(x_i)\eta_{j'}(x_{i'})\rangle_R\rangle_H &= \langle\eta(x_i)\langle\eta_{j'}(x_{i'})\rangle_R\rangle_H \\
&= \langle\eta(x_i)\times 0\rangle_H \\
&= 0.
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\sigma_S^2 &= \frac{1}{r^2n^2}(rn^2\sigma^2 + 0 + rn\langle\sigma^2(x_i)\rangle_H) \\
&= \frac{1}{r}\sigma^2 + \frac{1}{rn}\langle\sigma^2(x_i)\rangle_H.
\end{aligned}$$

■