Optimization of Control Parameters for Genetic Algorithms

JOHN J. GREFENSTETTE, MEMBER, IEEE

Abstract—**The task of optimizing a complex system presents at least two levels of problems for the system designer. First, a class of optimization algorithms must be chosen that is suitable for application to the system. Second, various parameters of the optimization algorithm need to be tuned for efficiency. A class of adaptive search procedures called genetic algorithms (GA) has been used to optimize a wide variety of complex systems. GA's are applied to the second level task of identifying efficient GA's for a set of numerical optimization problems. The results are validated on an image registration problem. GA's are shown to be effective for both levels of the systems optimization problem.**

I. Introduction

THE PROBLEM of dynamically controlling a complex process often reduces to a numerical function optimization problem. Each task environment for the process defines a performance response surface which must be explored in general by direct search techniques in order to locate high performance control inputs (see Fig. 1).

If the response surface is fairly simple, conventional nonlinear optimization or control theory techniques may be suitable. However, for many processes of interest, e.g. computer operating systems or system simulation programs, the response surface is difficult to search, e.g., a high-dimensional, multimodal, discontinuous, or noisy function of the control inputs. In such cases, the choice of optimization technique may not be obvious. Even when an appropriate class of optimization algorithms is available, there are usually various parameters that must be tuned, e.g., the step size in a variable metric technique. Often the choice of parameters can have significant impact on the effectiveness of the optimization algorithm [8]. The problem of tuning the primary algorithm represents a secondary, or metalevel, optimization problem (see Fig. 2).

This work attempts to determine the optimal control parameters for a class of global optimization procedures called genetic algorithms (GA's). The class of GA's is distinguished from other optimization techniques by the use of concepts from population genetics to guide the search. However, like other classes of algorithms, GA's differ from one another with respect to several parameters and strategies. This paper describes experiments that search a parameterized space of GA's in order to identify efficient GA's for the task of optimizing a set of numerical functions. This search is performed by a metalevel GA. Thus GA's are shown to be suitable for both levels of the system optimization problem.

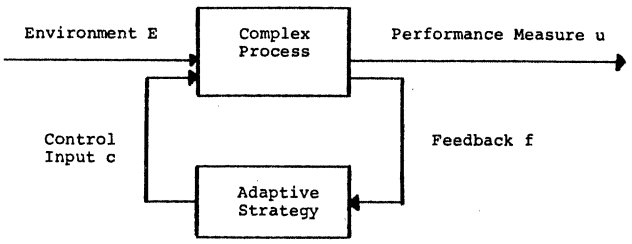

Fig. 1 One-level adaptive system model.

The remainder of this paper is organized as follows: Section II contains a brief overview of GA's and a summary of previous work. Section III describes the design of experiments which test the performance of GA's as meta-level optimization strategies. The experimental results appear in Section IV. A validation study is presented in Section V. The conclusions are summarized in Section VI.

II. Overview of Genetic Algorithms

Suppose we desire to optimize a process having a response surface u, which depends on some input vector x. It is assumed that no initial information is available concerning the surface u, but that a black box evaluation procedure can be invoked to compute the scalar function $u(x)$. The state of the art in such situations is to perform some sort of random search, perhaps combined with local hill-climbing procedures [5], [9]. Genetic algorithms are global optimization techniques that avoid many of the shortcomings exhibited by local search techniques on difficult search spaces.

A GA is an iterative procedure which maintains a constant-size population $P(t)$ of candidate solutions. During each iteration step, called a *generation*, the structures in the current population are evaluated, and, on the basis of those evaluations, a new population of candidate solutions is formed (see Fig. 3.)

The initial population $P(0)$ can be chosen heuristically or at random. The structures of the population $P(t+1)$ are chosen from $P(t)$ by a randomized selection procedure that ensures that the expected number of times a structure is chosen is approximately proportional to that structure's

Manuscript received March 21, 1984; revised August 28, 1985. This work was supported in part by a Fellowship from the Vanderbilt University Research Council and by the National Science Foundation under Grant MCS-8305693.

The author is with the Computer Science Department, Vanderbilt University, Nashville, TN 37235, USA.

IEEE Log Number 8406073.

0018-9472/86/0100-0122$01.00 ©1986 IEEE

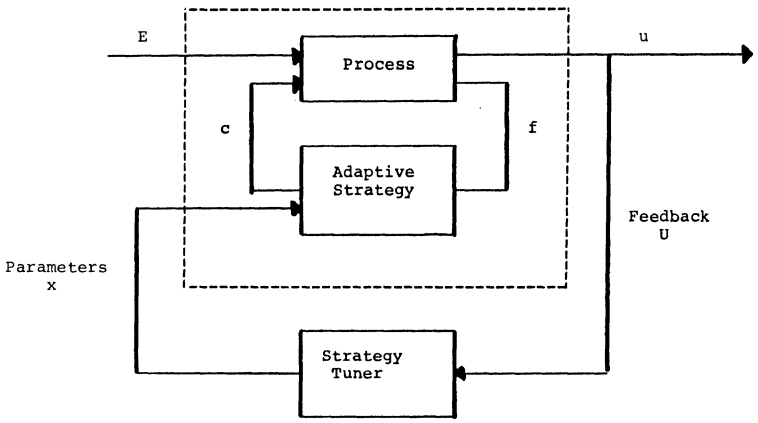

Fig. 2 Two-level adaptive system model.

```
t <- 0;
initialize P(t); -- P(t) is the population at time t
evaluate P(t);
while (termination condition not satisfied) do
begin
        t <- t+1;
        select P(t);
        recombine P(t);
        evaluate P(t);
end;
```

Fig. 3 Skeleton of a genetic algorithm.

performance relative to the rest of the population. In order to search other points in the search space, some variation is introduced into the new population by means of idealized *genetic recombination operators*. The most important recombination operator is called *crossover*. Under the crossover operator, two structures in the new population exchange portions of their internal representation. For example, if the structures are represented as binary strings, crossover can be implemented by choosing a point at random, called the crossover point, and exchanging the segments to the right of this point. Let x_1 = 100:01010 and x_2 = 010:10100, and suppose that the crossover point has been chosen as indicated. The resulting structures would be y_1 = 100:10100 and y_2 = 010:01010.

Crossover serves two complementary search functions. First, it provides new points for further testing within the hyperplanes already represented in the population. In the above example, both x_1 and y_1 are representatives of the hyperplane 100#####, where the # is a "don't care" symbol. Thus, by evaluating y_1, the GA gathers further knowledge about this hyperplane. Second, crossover introduces representatives of new hyperplanes into the population. In the previous example, y_2 is a representative of the hyperplane #1001###, which is not represented by either parent structure. If this hyperplane is a high-performance area of the search space, the evaluation of y_2 will lead to further exploration in this subspace. Each evaluation of a structure of length L contributes knowledge about the performance of the 2^L hyperplanes represented by that structure. The power of GA's derives largely from their ability to exploit efficiently this vast amount of accumulating knowledge by means of relatively simple selection mechanisms [17]. Termination of the GA may be triggered by finding an acceptable approximate solution, by fixing the total number of structure evaluations, or some other application dependent criterion. For a more thorough introduction to GA's [7], [17].

As stated above, GA's are essentially unconstrained search procedures within the given representation space. Constraints may be handled indirectly through penalty functions. A more direct way to incorporate constraints has been proposed by Fourman [12], who treats the structures in the population as lists of consistent constraints for VLSI layout problems.

Genetic algorithms have not enjoyed wide recognition, possibly due to a misconception that GA's are similar to early "evolutionary programming" techniques [11], which rely on random mutation and local hill-climbing. The basic concepts of GA's were developed by Holland [17] and his students [1], [2], [4], [6], [13], [15], [19]. These studies have produced the beginnings of a theory of genetic adaptive search. For example, an application of gambler's ruin theory to the allocation of trials to the hyperplanes of the search space shows that genetic techniques provide a near-optimal heuristic for information-gathering in complex search spaces [6], [17]. Bethke [1] provides theoretical characterizations of problems which may be especially well-suited or especially difficult for GA's [1]. In addition, a number of experimental studies show that GA's exhibit impressive efficiency in practice. While classical gradient search techniques are more efficient for problems which satisfy tight constraints, GA's consistently outperform both gradient techniques and various forms of random search on more difficult (and more common) problems, such as optimizations involving discontinuous, noisy, high-dimensional, and multimodal objective functions. GA's have been applied to various domains, including combinatorial optimization [12], [16], image processing [10], pipeline control systems [15], and machine learning [2], [18], [25].

III. Experimental Design

We now describe experiments which attempted to optimize the performance of GA's on a given set of function optimization problems. These experiments were designed to search the space of GA's defined by six control parameters, and to identify the optimal parameter settings with

respect to two different performance measures. The searches for the optimal GA's were performed by GA's, which demonstrates the efficiency and power of GA's as metalevel optimization techniques. A metalevel GA could similarly search any other space of parameterized optimization procedures.

A. The Space of Genetic Algorithms

Holland [17] describes a fairly general framework for the class of GA's. There are many possible elaborations of GA's involving variations such as other genetic operators, variable sized populations, etc. This study is limited to a particular subclass of GA's characterized by the following six parameters.

1) Population Size (N): The population size affects both the ultimate performance and the efficiency of GA's. GA s generally do poorly with very small populations [22], because the population provides an insufficient sample size for most hyperplanes. A large population is more likely to contain representatives from a large number of hyperplanes. Hence, the GA's can perform a more informed search. As a result, a large population discourages premature convergence to suboptimal solutions. On the other hand, a large population requires more evaluations per generation, possibly resulting in an unacceptably slow rate of convergence. In the current experiments, the population size ranged from 10 to 160 in increments of 10.

2) Crossover Rate (C): The crossover rate controls the frequency with which the crossover operator is applied. In each new population, $C * N$ structures undergo crossover. The higher the crossover rate, the more quickly new structures are introduced into the population. If the crossover rate is too high, high-performance structures are discarded faster than selection can produce improvements. If the crossover rate is too low, the search may stagnate due to the lower exploration rate. The current experiments allowed 16 different crossover rates, varying from 0.25 to 1.00 in increments of 0.05.

3) Mutation Rate (M): Mutation is a secondary search operator which increases the variability of the population. After selection, each bit position of each structure in the new population undergoes a random change with a probability equal to the mutation rate M. Consequently, approximately $M * N * L$ mutations occur per generation. A low level of mutation serves to prevent any given bit position from remaining forever converged to a single value in the entire population. A high level of mutation yields an essentially random search. The current experiments allowed eight values for the mutation rate, increasing exponentially from 0.0 to 1.0.

4) Generation Gap (G): The generation gap controls the percentage of the population to be replaced during each generation. That is $N*(1 - G)$ structures of $P(t)$ are chosen (at random) to survive intact in $P(t + 1)$. A value of $G = 1.0$ means that the entire population is replaced during each generation. A value of $G = 0.5$ means that half of the structures in each population survive into the next generation. The current experiments allowed G to vary between 0.30 and 1.00, in increments of 0.10.

5) Scaling Window (W): When maximizing a numerical function $f(x)$ with a GA, it is common to define the performance value $u(x)$ of a structure x as $u(x) = f(x) - f_{\min}$, where $f_{\min}$ is the minimum value that $f(x)$ can assume in the given search space. This transformation guarantees that the performance $u(x)$ is positive, regardless of the characteristics of $f(x)$. Often, $f_{\min}$ is not available *a priori*, in which case it is reasonable to define $u(x) = f(x) - f(x_{\min})$, where $f(x_{\min})$ is the minimum value of any structure evaluated so far. Either definition of $u(x)$ has the unfortunate effect of making good values of x hard to distinguish. For example, suppose $f_{\min} = 0$. After several generations, the current population might contain only structures x for which $105 < f(x) < 110$. At this point, no structure in the population has a performance which deviates much from the average. This reduces the selection pressure toward the better structures, and the search stagnates. One solution is to define a new parameter $f'_{\min}$ with a value of say, 100, and rate each structure against this standard. For example, if $f(x_i) = 110$ and $f(x_j) = 105$, then $u(x_i) = f(x_i) - f'_{\min} = 10$, and $u(x_j) = f(x_j) - f'_{\min} = 5$; the performance of x_i now appears to be twice as good as the performance of x_j.

Our experiments investigated three scaling modes, based on a parameter called the scaling window W. If $W = 0$, then scaling was performed as follows: $f'_{\min}$ was set to the minimum $f(x)$ in the first generation. For each succeeding generation, those structures whose evaluations were less than $f'_{\min}$ were ignored in the selection procedure. The $f'_{\min}$ was updated whenever all the structures in a given population had evaluations greater than $f'_{\min}$. If $0 < W < 7$, then we set $f'_{\min}$ to the least value of $f(x)$ which occurred in the last W generations. A value of $W = 7$ indicated an infinite window (i.e., no scaling was performed).

6) Selection Strategy (S): The experiments compared two selection strategies. If $S = P$, a *pure selection* procedure was used, in which each structure in the current population is reproduced a number of times proportional to that structure's performance. If $S = E$, an *elitist strategy* was employed. First, pure selection is performed. In addition, the elitist strategy stipulates that the structure with the best performance always survives intact into the next generation. In the absence of such a strategy, it is possible for the best structure to disappear, due to sampling error, crossover, or mutation.

We denote a particular GA by indicating its respective values for the parameters N, C, M, G, W, and S. Early work by De Jong [6] suggests parameter settings which have been used in a number of implementations of genetic algorithms. Based on De Jong's work, we define the *standard GA* as $\mathrm{GA}_S = \mathrm{GA}(50, 0.6, 0.001, 1.0, 7, E)$. The Cartesian product of the indicated ranges for the six parameters (N, C, M, G, W, S) defines a space of 2^{18} GA's. In some cases, it is possible to predict how variations of a single parameter will affect the performance of the GA's, assuming that all other parameters are kept fixed [6].

However, it is difficult to predict how the various parameters interact. For example, what is the effect of increasing the population size, while lowering the crossover rate? The analytic optimization of this space is well beyond our current understanding of GA's. It is also clear that an exhaustive enumeration of the space is infeasible. Our approach was to apply a metalevel GA's to the problem of identifying high-performance GA's. Each structure in the population of the meta-level GA consisted of an 18-bit vector which identified a particular GA. The performance of each GA was measured during the performance of a series of function optimization tasks. The meta-level GA used this information to conduct a search for high-performance algorithms.

B. Task Environment

Each GA was evaluated by using it to perform five optimization tasks, one for each of five carefully selected numerical test functions. As a results of these optimization tasks, the GA was assigned a value according to one of the performance measures explained below. The functions comprising the task environment have been studied in previous studies of GA's [3], [6] and included functions with various characteristics, including discontinuous, multidimensional, and noisy functions. Table I gives a brief description of the test functions.

C. Performance Measures

Two performance metrics for adaptive search strategies were considered, *online performance* and *offline performance* [6]. The on-line performance of a search strategy s on a response surface e is defined as follows:

$$U_e(s,T) = \text{ave}_t(u_e(t)), \qquad t = 0,1,\cdots,T$$

where $u_e(t)$ is the performance of the structure evaluated at time t. That is, online performance is the average performance of all tested structures over the course of the search.

The offline performance of a search strategy s on a response surface e is defined as follows:

$$U_e^*(s,T) = ave_t(u_e^*(t)), \qquad t = 0,1,\cdots,T$$

where $u_e^*(t)$ is the best performance achieved in the time interval $[0, t]$. Offline performance is the relevant measure when the search can be performed offline (e.g., via a simulation model), while the best structure so far is used to control the online system.

In order to measure global robustness, corresponding performance measures are defined for the entire set of response surfaces E:

$$U_E(s,T) = 100.0 * \text{ave}_{e \text{ in } E}(U_e(s,T)/U_e(\text{rand},T)),$$

$$U_E^*(s,T) = 1000 * \text{ave}_{e \text{ in } E}(U_e^*(s,T)/U_e^*(\text{rand},T)),$$

where U_e (rand, T) and U_e^*(rand, T) are on-line and offline performance, respectively, of pure random search on response surface e. As normalized, U_E and U_E^* for random search will be 100.0, while U_E and U_E^* for more effective search strategies will be correspondingly lower (for minimization problems).

TABLE I
FUNCTIONS COMPRISING THE TEST ENVIRONMENT

Function	Dimensions	Size of Space	Description
$f1$	3	1.0×10^9	parabola
$f2$	2	1.7×10^6	Rosenbrock's saddle [23]
$f3$	5	1.0×10^{15}	step function
$f4$	30	1.0×10^{72}	quartic with noise
$f5$	2	1.6×10^{10}	Shekel's foxholes [24]

D. Experimental Procedures

Two experiments were performed, one to optimize on-line performance and one to optimize offline performance. For each experiment, the procedure for obtaining the optimum GA was as follows.

1) One thousand GA's were evaluated, using a metalevel GA to perform the search through the space of GA's defined by the six GA parameters. Each evaluation comprised running one GA against each of the five test functions for 5000 function evaluations and normalizing the result with respect to the performance of random search on the same function. The metalevel GA started with a population of 50 randomly chosen GA's and used the standard parameter settings, i.e., GA(50, 0.6, 0.001, 1.0, 7, E). Past experience has shown that these parameters yield a fairly good search for a variety of problems, and so this was the natural choice for the meta-level.

2) Since GA's are randomized algorithms, the performance of a GA during a single trial in the metalevel experiment represents a sample from a distribution of performances. Therefore, it was decided that the GA's showing the best performances during step 1 would be subjected to more extensive testing. Each of the 20 best GA's in step 1 was again run against the task environment, this time for five trials for each test function, using different random number seeds for each trial. The GA which exhibited the best performance in this step was declared the winner of the experiment.

IV. RESULTS

A. Experiment 1—Online Performance

The first experiment was designed to search for the optimal GA with respect to online performance on the task environment. Fig. 4 shows the average online performance for the 50 GA's in each of the 20 generations of experiment 1. Recall that the overall scores for random search on the task environment is 100.0. From the initial data point in Fig. 4, we can estimate that the average online performance of all GA's in our search space is approximately 56.6, or about 43.4 percent better than random search. Fig. 4 shows that the final population of GA's had significantly better performance than the average GA.

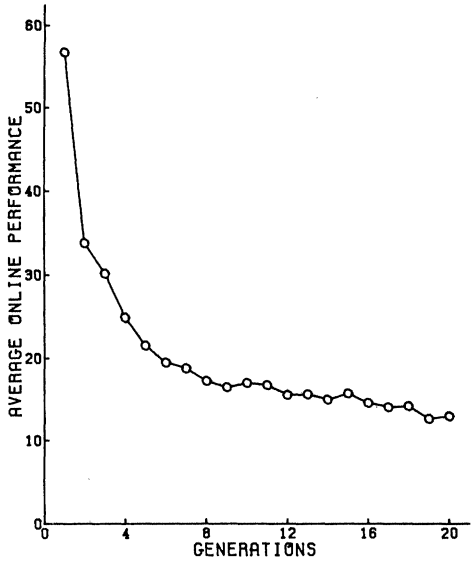

Fig. 4 Experiment 1.

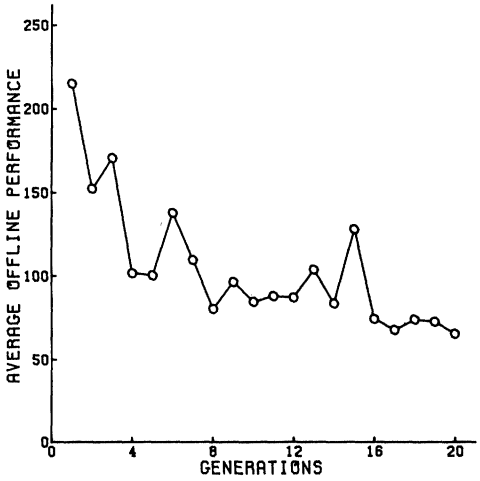

Fig. 5 Experiment 2.

This experiment identified GA_1 = GA(30, 0.95, 0.01, 1.0, 1, E) as the optimal GA with respect to online performance. In an extended comparison, GA_1 showed a 3.09 percent improvement (with respect to the baseline performance of random search) over GA_S on the task environment. This represents a small but statistically significant improvement over the expected online performance of GA_S. The performance improvement between GA_S and GA_1 can be attributed to an interaction among a number of factors. First, GA_1 uses a smaller population, which allows many more generations within a given number of trials. For example, on functions in the task environment, GA_1 iterated through an average of twice as many generations as GA_S. Second, GA_S uses an infinite window, i.e., no scaling is performed. GA_1 uses a small window (one generation), which resulted in a more directed search. These two factors are apparently balanced by the significantly increased crossover rate and mutation rate in GA_1. A higher crossover rate tends to disrupt the structures selected for reproduction at a high rate, which is important in a small population, since high performance individuals are more likely to quickly dominate the population. The higher mutation rate also helps prevent premature convergence to local optima.

B. Experiment 2—Offline Performance

The second experiment was designed to search for the optimal GA with respect to offline performance on the task environment. Fig. 5 shows that the average offline performance of all GA's (214.8) appears to be much worse than the average offline performance of random search (100.0). This finding verifies the experience of many practitioners that GA's can prematurely converge to suboptimal solutions when given the wrong control parameters. For example, if a very small population size is used (i.e., N = 10), the number of representatives from any given hyperplane is so small that the selection procedure has insufficient information to properly apportion credit to the hyperplanes represented in the population. As a result, a relatively good structure may overrun the entire population in a few generations. Unless the mutation rate is high, the GA will quickly converge to a suboptimal solution. In contrast, random search will usually locate at least one high performance point within the first thousand trials, leading to relatively good offline performance. That is, random search is a fairly tough competitor for search strategies when the goal is good offline performance. It is encouraging that many GA's perform significantly better than random search with respect to the offline performance measure.

This experiment identified GA_2 = GA(80, 0.45, 0.01, 0.9, 1, P) as the optimal GA with respect to offline performance. In an extended comparison, GA_2 showed a 3.0 percent performance improvement over GA_S on the task environment. Because of the high variance shown by GA's with respect to offline performance, this does not represent a statistically significant difference between GA_2 and GA_S. There are several interesting difference between GA_2 and GA_S. With a larger population and higher mutation rate, the population will tend to contain more variety, thus increasing the random aspects of the GA. The slightly lower generation gap also tends to reduce the effects of selection, resulting in a less focused search. These aspects are balanced by the lower crossover rate and the small scaling window which tend to enhance the selective pressure.

C. General Observations

Besides suggesting optimal GA's, the above experiments also provide performance data for 2000 GA's with various parameter settings. Given that these are not independent samples from the space of GA's, it is rather difficult to make valid statistical inferences from this data. Nevertheless the data does suggest some regularities that might warrant further studies.

The experimental data confirms several observations first made by De Jong on the basis of a relatively small number of experiments [6]. For example, mutation rates above 0.05 are generally harmful with respect to online performance,

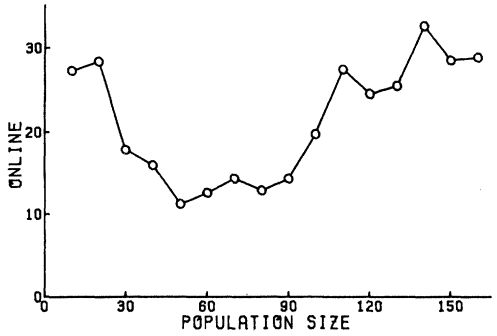

Fig. 6 Average online performance of various population sizes according to Experiment 1.

with performance approaching that of random search with rates above 0.1 regardless of the other parameter settings. The absence of mutation is also associated with poorer performance, which suggests that mutation performs an important service in refreshing lost values. Best on-line performance can be obtained with a population size in the range of 30–100 structures as shown by Fig. 6, which plots the average online performance as a function of population size, ignoring those GA's with mutation rates above 0.05.

A similar graph can be obtained for offline performance except that range for the best population size is 60–110 structures. A large generation gap generally improves performance, as does the elitist selection strategy.

The performance data also suggests other regularities that have not been previously noted. First, the adoption of a small scaling window (1–5 generations) is associated with a slight improvement in both online and offline performance. This is reasonable since scaling enhances the pressure of selection in later stages of the search. In small populations (20 to 40) structures, good online performance is associated with either a high crossover rate combined with a low mutation rate or a low crossover rate combined with a high mutation rate. For mid-sized populations (30 to 90 structures), the optimal crossover rate appears to decrease as the population size increases. For example, among the best 10 percent of all GA's with population size 30, the average crossover rate was 0.88. The best crossover rate decreases to 0.50 for population size 50 and to 0.30 for population size 80. This is reasonable since, in smaller populations, crossover plays an important role in preventing premature convergence. In summary, the performance of GA's appear to be a nonlinear function of the control parameters. However, the available data is too limited to confirm or disconfirm the existence of discontinuities or multiple local optima in the performance space of GA's. It would be interesting to compare the performance of other nonlinear search techniques in optimizing the performance of GA's.

V. Validation

In order to validate the experimental results, the three algorithms GA_S, GA_1 and GA_2 were applied to an optimization problem which was not included in the experimental

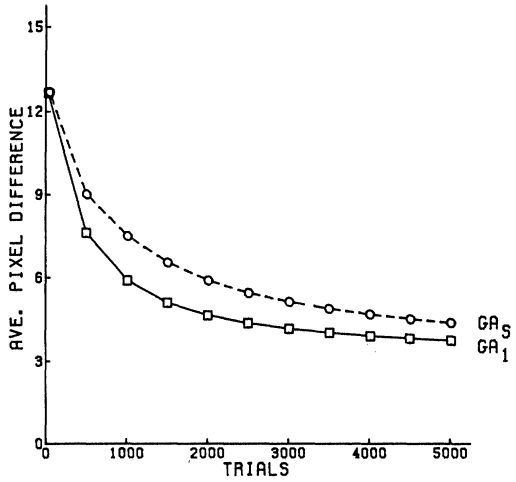

Fig. 7 Online performance of GA_1 and GA_S on an image registration task.

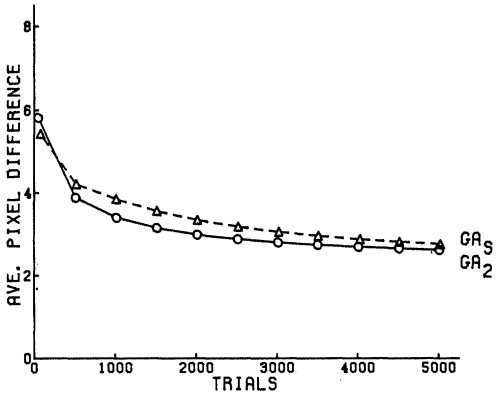

Fig. 8 Offline performance of GA_2 and GA_S on an image registration task.

task environment. The validation task was the following image registration problem: In order to compare two gray-level images of a scene taken at different times or from different vantage points, it is often necessary to determine a transformation, or *registration*, which will map one image, the original image, into another, the target image. This registration problem is important in such diverse fields as aerial photography and medical imaging [20], [21]. One approach to the image registration problem [10] is to define a parameterized class of transformations and to apply a GA to the task of searching this class for an optimal transformation, i.e., a transformation which maps the original image into the target image. The response surface for a given pair of images is a function of the transformation and corresponds to the average gray-level differences between corresponding pixels in the transformed original image and the target image. This class of problems appears to be difficult for conventional nonlinear programming algorithms due to the inevitable presence of many local minima [14].

Experiments were conducted to compare the effectiveness of the algorithms GA_S, GA_1, and GA_2 for a sample registration problem consisting of a pair of carotid artery images in which patient motion produced significant mo-

tion artifacts. Each experiment consisted of five runs of a GA on the pair of images, each run evaluating 5000 candidate transformations. Fig. 7 compares the online performance of GA_S with GA_1. GA_1 shows a small but statistically significant improvement in online performance over GA_S on this problem. Fig. 8 shows that GA_2 produces no significant difference in offline performance over GA_S. These results are consistent with the results in Experiments 1 and 2, and indicate that those results may be generally applicable to other optimization problems.

VI. Conclusion

Experiments were performed to search for the optimal GA's for a given set of numerical optimization problems. Previous experiments show that the standard GA, GA_S, outperforms several classical optimization techniques on the task environment. Thus, one goal of our experiments was to identify GA's which are at least as good as GA_S. Both experiments succeeded in identifying control parameters settings that optimize GA's with respect to the described performance metrics. The experimental data also suggests that, while it is possible to optimize GA control parameters, very good performance can be obtained with a range of GA control parameter settings.

The present approach is limited in several ways. First, it was necessary to choose a particular parameterized subclass of GA's to explore. In particular, we have neglected other recombination operators such as multipoint crossover and inversion and other strategies such as the inclusion of a "crowding factor" [6]. Second, the GA's we considered were essentially unconstrained optimization procedures. As previously noted, there are ways to incorporate constraints into GA's but it remains for future research to determine how the presence of constraints affects the optimal control parameters. Finally, the metalevel experiments represent a sizable number of CPU hours. It is encouraging that the results appear to be applicable to a wide class of optimization problems.

An alternative approach to the optimization of GA's would be to enable the GA to modify its own parameters dynamically during the search. However, for many optimization problems the number of evaluations which can be performed in a reasonable amount of time would not allow the GA enough evaluations to modify its search techniques to any significant degree. Therefore, the experiments described above are important in that they identify approximately optimal parameter settings for the two performance measures considered. The data also suggests several new tradeoffs among the control parameters which may lead to further theoretical insights concerning the behavior of genetic algorithms.

References

[1] A. D. Bethke, "Genetic algorithms as function optimizers," Ph. D. thesis, Dept. Computer and Communication Sciences, Univ. of Michigan, 1981.

[2] L. B. Booker, "Intelligent behavior as an adaptation to the task environment," Ph. D. thesis, Dept. Computer and Communication Sciences, Univ. of Michigan, Feb. 1982.

[3] A. Brindle, "Genetic algorithms for function optimization," Ph. D. thesis, Computer Science Dept., Univ. of Alberta, 1981.

[4] D. J. Cavicchio, "Adaptive search using simulated evolution," Ph.D. thesis, Dept. Computer and Communication Sciences, Univ. of Michigan, 1970.

[5] L. Cooper and D. Steinberg, *Methods of Optimization*. Philadelphia: W. B. Saunders, 1970.

[6] K. A. DeJong, *Analysis of the behavior of a class of genetic adaptive systems*, Ph.D. thesis, Dept. Computer and Communication Sciences, Univ. of Michigan, 1975.

[7] ——, "Adaptive system design: A genetic approach," *IEEE Trans. Syst., Man, Cyber.* vol. SMC-10, no. 9, pp. 566–574, Sept. 1980.

[8] L. C. W. Dixon, "The choice of step length, a crucial factor in the performance of variable metric algorithms," in *Numerical Methods for Nonlinear Optimization*, F. A. Lootsma, Ed. New York: Academic, 1972.

[9] W. Farrell, *Optimization Techniques for Computerized Simulation*. Los Angeles: CACI, 1975.

[10] J. M. Fitzpatrick, J. J. Grefenstette, and D. Van Gucht, "Image registration by genetic search," in *Proc. of IEEE Southeastcon '84*, pp. 460–464, Apr. 1984.

[11] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley and Sons, 1966.

[12] M. P. Fourman, "Compaction of symbolic layout using genetic algorithms," *Proc. Intl. Conf. on Genetic Algorithms and their Applications*, pp. 141–153, July 1985.

[13] D. R. Frantz, *Non-linearities in genetic adaptive search*, Ph.D. thesis, Dept. Computer and Communication Sciences, Univ. of Michigan, 1972.

[14] W. Frei, T. Shibata, and C. C. Chen, "Fast matching of non-stationary images with false fix protection," *Proc. 5th Intl. Conf. Patt. Recog.*, vol. 1, pp. 208–212, IEEE, 1980.

[15] D. Goldberg, "Computer-aided gas pipeline operation using genetic algorithms and rule learning," Ph.D. thesis, Dept. Civil Eng., Univ. of Michigan, 1983.

[16] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proc. Intl. Conf. Genetic Algorithms and their Applications*, pp. 160–168, July 1985.

[17] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. Michigan, Ann Arbor, MI, 1975.

[18] ——, "Escaping brittleness," in *Proc. Int. Machine Learning Workshop*, pp. 92–95, June 1983.

[19] R. B. Hollstien, *Artificial Genetic Adaptation in Computer Control Systems*, Ph.D. Thesis, Dept., Computer and Communication Sci ences, Univ. of Michigan, 1971.

[20] R. A. Kruger and S. J. Riederer, *Basic Concepts of Digital Subtraction Angiography*. Boston: G. K. Hall, 1984.

[21] James J. Little, "Automatic registration of landsat MSS images to digital elevation models," in *Proc. IEEE Workshop Computer Vision: Representation and Control*, pp. 178–184, 1982.

[22] E. Pettit and K. M. Swigger, "An analysis of genetic-based pattern tracking," in *Proc. National Conf. on AI, AAAI 83*, pp. 327–332, 1983.

[23] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *Computer J.*, vol. 3, pp. 175–184, Oct. 1960.

[24] J. Shekel, "Test functions for multimodal search techniques," in *Fifth Ann. Princeton Conf. Inform. Sci. Syst.*, 1971.

[25] S. F. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proc. of 8th IJCAI* 1983.