12-1-1993

Genetic Algorithms in Noisy Environments

T. W. THEN
Purdue University School of Electrical Engineering

EDWIN K. P. CHONG
Purdue University School of Electrical Engineering

GENETIC ALGORITHMS IN NOISY ENVIRONMENTS

T. W. THEN
E. K. P. CHONG

TR-EE 93-36
NOVEMBER 1993

SCHOOL OF ELECTRICAL ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Genetic Algorithms in Noisy Environments

T. W. THEN and EDWIN K. P. CHONG *

School of Electrical Engineering, Purdue University

West Lafayette, IN 47907-1285

Abstract

Genetic Algorithms (GA) have been widely used in the areas of searching, function optimization, and machine learning. In many of these applications, the effect of noise is a critical factor in the performance of the genetic algorithms. While it has been shown in previous studies that genetic algorithms are still able to perform effectively in the presence of noise, the problem of locating the global optimal solution at the end of the search has never been effectively addressed. Furthermore, the best solution obtained by GA often does not coincicle with the optimal solution for the problem when noise is present. In this report, we describe a modified GA for dealing with noisy environments. We use an optimal solution list to keep a dynamic record of the optimal solutions that have been found during the course of evolution of the population of noisy solutions. In addition, we also vary the population size and sampling rate to achieve further improvements. We demonstrate the performance of our scheme via a simple function optimization problem using genetic algorithm in a noisy environment. Our results show that the optimal solution list is able to provide a small solution set that provides near optimal solutions obtainable in the absence of noise. Our scheme is also easily implemented in practice with the addition of a simple optimal solution list and minor changes to the selection and evaluation phases of an existing GA implementation.

"**Research partially supported by a grant from the Purdue NSF Engineering Research Center for Intelligent Manufacturing**

1 Introduction

Although genetic algorithms have been applied in a variety of domains, including image processing, machine learning, combinatorial optimization, neural network design, robotics, and function optimization [1, 2], there is still a large class of practical problems where genetic algorithms have not been applied simply because these problems require the evaluation of thousands of candidate solutions which can prove to be very computationally intensive and expensive. Nevertheless, genetic algorithm has proven to be very effective in large and complex search space (e.g., high-dimensional, discontinuous spaces with many local optima), even more so than many of the traditional random search and local search techniques [3]. As such, it would be advantageous if we can apply genetic algorithms to those large practical problems to reduce the amount of computation. In fact, for many such problems, it is sufficient to evaluate the candidate solutions approximately using statistical sampling techniques. Motivated by this, Fitzpatrick and Grefenstette [5] have established improved performance resulting from decreasing effort in approximating function evaluations and increasing the number of iterations of the genetic algorithm. The same authors have also considered the effects of varying both the population size and the sampling rate in a later work [4]. In both studies, the overall performance of the genetic algorithm has been shown to be markedly superior even in a noisy environment. However, the question of which candidate solution in the last generation or in any of the previous generation for that matter is the optimal solution remains unanswered.

In this report we propose the use of a simple optimal solution list and an appropriate balance between population size and sampling rate to be incorporated with existing genetic algorithm implementations in the presence of noise. The purpose of the list is to keep a dynamic record of potential solutions found during the course of the GA run so as to overcome the problem of getting an otherwise inaccurate optimal solution when using a regular genetic algorithm without the list. We will demonstrate this inadequacy using a function optimization problem in the presence of gaussian noise. We show that the optimal solution obtained using a simple GA without the list frequently gives the wrong optimal solution in a

noisy environment. Using the same function optimization problem, we demonstrate that the genetic algorithm with an optimal solution list provides a more consistent and accurate optimal solution even in the presence of an increasingly noisy environment. Further evidence is provided in which a dynamic balance in the amount of effort spent on evaluating each candidate solution and the number of candidate solutions evaluated during each iteration of the genetic algorithm is shown to improve the results even further. The proposed scheme can be implemented simply by adding an additional list to existing GA implementations and minor changes to the selection and evaluation phase of the GA implementation.

The remainder of this report is organized as follows. In Section 2, we describe the basic principles of genetic algorithms. In Section 3, we present the basic structure of our proposed scheme. We will demonstrate the effectiveness of our scheme on a function optimization problem in a noisy environment in Section 4 and 5. Finally, we conclude in Section 6.

Review Of Genetic Algorithms

Genetic algorithms are probabilistic search techniques based on the principle of population genetics. This class of algorithms can be classified as a subclass of a larger class of algorithms based on the concept of evolution. Since its conception in the late 1960's and early 1970's (as a result of the work of John Holland at the University of Michigan), a inyraid of studies have been conducted on almost every aspect of the algorithm, giving rise to a large volume of literature on this algorithm. Concurrently, there has also been widespread applications of genetic algorithms to various practical problems from different domains. Recently, the algorithm has received increase attention as a result of its successfulness in solving many difficult problems. The following is a brief discussion of the basic concepts underlining the workings of the genetic algorithm.

We shisll proceed with the discussion using a simple genetic algorithm. The genetic algorithm maintains a population $P(t)$ of N candidate solutions $\{x_1, x_2, \ldots, x_N\}_t$ chosen from the solution space. During iteration t, the population of candidate solutions undergoes a process of selection by fitness and evolution to locate the optimal solution. This proceeds

for the duration of the search until the termination condition is satisfied. It is through this process of selection and evolution that the population of candidate solutions or chromosomes improves itnd converges towards the global optima. Because the genetic algorithm performs a multi-directional search of the solution space by maintaining a fixed size population of solutions as opposed to a single candidate solution at any given iteration, the search is very efficient. This gives rise to the implicit parallelism of the genetic algorithm as noted by Holland [6]. The basic structure of a simple GA is shown below

```
GeneticAlgorithm
{
    t = O
    initialize P(t)
    evaluate P(t)
    while (not termination condition)
    {
        t = t + 1
        select P(t) from P(t - 1)
        evolve P(t)
        evaluate P(t)
    }
}
```

We will proceed to discuss the algorithm in detail.

The first phase of a simple genetic algorithm is the encoding of the solution space into a suitable representation. Traditionally, as used in the original representation in Holland's work, this has taken the form of binary strings, that is, strings of 1's and 0's. Using this scheme of .representation,the various components of a solution are encoded into binary strings which are then concatenated to form a single binary string called a *chromosome.* Although binary representation has been very successful in encoding solutions for many problems, there are still limitations in that not all solutions, especially those of pratical problems, can be encoded in this manner. As a result, other forms of representation, including real number representation, have been explored and studied. In many cases, these forms of representation have proven to be very effective in encoding the solutions. However, many of these have not been formally analyzed. Thus for simplicity, we shall use the binary representation in our

discussion.

Once it suitable representation has been chosen, the next phase is to :initialize the first population of chromosomes. This is usually done by a random generation of the binary strings representing these chromosomes. In this way, a uniform representation of the solution space in the very first generation is ensured so that the algorithm will not converge prematurely to a local optima.

After the initial population of chromosomes has been formed, it will undergo a process of evolution. During each iteration t of the process, each candidate solution x; is evaluated by computing f(x;) which would include the objective function as well as other problem constraints. This provides a measure of fitness of the given candidate solution for the given problem. When the whole population has been evaluated, a new populaition of candidate solutions is then formed in two stages. In the first stage, the chromosomes are chosen stochastically to form the parents for the next population based on their relative fitness. In practice, the chromosomes of the present population are replicated according to their relative fitness by a stochastic procedure such that the number of replications for each chromosome x_i is on the average proportional to

$$\frac{f(x_i)}{\overline{F}(t)} \tag{1}$$

where $f(x_i)$ is the evaluated fitness or performance of the given chromosome x_i and $\overline{F}(t)$ is the average fitness of the population at the t iteration. In this way, the chromosomes that perform better than average will be chosen several times for the next generation while those that perform poorly are replicated less or not even at all. Thus, the better-performing chromosomes will gradually occupy more and more of the population with each passing iteration as a result of the selection pressure. However, this alone is insufficient to locate the global solution or local optima.

Just as in population genetic, there must be some ways for the population to evolve by introducing variations to the population. This is done during the second phase which is also called the reproduction phase. In genetic algorithm, this is achieved by two basic operators, namely the crossover and the mutation operators. Crossover allows us to mate potential chromosomes to combine the quality components (also called *genes*) from each

parent. This can be done by combining genetic materials from two parent; chromosomes to produce two new child chromosomes. For each pair of parent chromosomes, a random point is selected on both chromosomes (the same point). The chromosomes would then combine the corresponding segments between the crossover point so that each child has the first segment of one parent and the second segment of the other parent. In this way, the two chromosomes

abcdefg and ***ABCDEFG***

would become

$abcDEFG$ and $ABCdefg$

after crossover at the crossover point between the third and fourth gene. Any two parent chromosomes would undergo crossover with a probability of p_c. The crossover operator is the key operator of evolution of the genetic algorithm. After crossover, numerous alternatives dealing with the resulting strings can be implemented.

The mutation operator serves to exploit a solution by searching around a candidate solution to locate a better solution. This can be done by randomly changing each of the component bit of the chromosomes from 1 to 0 or 0 to 1 with a certain probability p_m. In most cases, only a few of the component bits are mutated since the probability of mutation is set at a very low value. This ensures that the mutation operator plays only a background role in the genetic algorithm as opposed to the crossover operator. After applying the two operators on the selected parent chromosomes, the next generation of chromosomes is formed and the whole process continues. During each iteration, the solution that has the best performance so far is recorded and at the end of the process, the final value recorded is the global optimal solution.

2.1 Theoretical Background

The power of genetic algorithm lies in the parallel search of the solution space. This is made possible by the efficient exploitation of the wealth of information thist the evaluation of the chromosomes provides. Specific configurations of the component values observed to contribute to the good performance of the chromosome are preserved and propagated through successive generations in a highly parallel fashion. These successful small configurations, in

turn, forrn the building blocks for the generation of larger configurations in subsequent generation, giving rise to an improvement of the population of candidate solutions as more and more successful configurations are combined together and replicated. This is the essence of the Building Block Hypothesis [7] which states that

Hypothesis 1 (Building Block Hypothesis) ***A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high-performace schemata, called the building blocks.***

The ability of the genetic algorithm to perform such an efficient search of the solution space is called the ***implicit parallelism*** of the genetic algorithm [6].

More specifically, consider a set of finite binary strings of length l. A natural method of representing the similarities of these strings is by the use of wildcards or don't care symbol (*) in those positions that are different, that is to say, in those positions that we are not interested. The structure formed in such a manner is called a schema or similarity template [7]. It encodes the similarity of a set of binary strings. For example, the schema $(*011100110)$ matches the following two strings

$$\{(0011100110), (1011100110)\},$$

while the schema $(*0*1100110)$ matches four strings

$$\{(0001100110), (0011100110), (1001100110), (1011100110)).$$

With this, we can see that every schema matches exactly 2^r strings where r is the number of don't cares in the schema. Furthermore, a string of length l can be represented by 2^l schemata (plural for schema).

Now, different schemata have different characteristics and the two basic schema properties are the idea of order and defining length. The order of a schema H, $o(H)$, refers to the number of fixed digits in the string. In other words, the order gives an indication of how specialized is the schema. For example, the following schemata

$$H_1 = **101**11$$
$$H_2 = 11***10*00$$
$$H_3 = *01*11001*$$

would have the following order

$$o(H_1) = 5, o(H_2) = 6, o(H_3) = 7,$$

with H_3 being the most specific.

The defining length of a schema H, $\delta(H)$, refers to the distance between the schema's first fixed digit and the last fixed digit in the string. For example, using the schemata defined above, the corresponding defining lengths are as follows,

$$\delta(H_1) = 10 - 4 = 6$$
$$\delta(H_2) = 10 - 1 = 9$$
$$\delta(H_3) = 9 - 2 = 7.$$

Next, the schema has another property, and that is the fitness of the schema H at a given iteration t, $f(H, t)$. This is given by the average fitness of all the strings that match the schema. For example, assuming that there are N strings $\{x_1, x_2, \ldots, x_N\}$ in the population that match the schema H at the t iteration, then

$$f(H,t) = \frac{f(x_1) + f(x_2) + \ldots + f(x_N)}{N} \tag{2}$$

During the process of selection as discussed earlier, the probability that a string will be selected depends on its relative fitness as compared to the rest of the population. This probability is given by equation (1). Let the number of strings matched by the schema H at the t^{th} iteration be $\mathcal{E}(H,t)$. Under the proportional reproduction and selection process, the proba'bility that an average string is matched by the schema H is equal to $f(H,t)/F(t)$ where $F(t)$ is the total fitness of the population. Given that the number of strings matching the schema H at the t^{th} iteration is $\mathcal{E}(H,t)$, the number of strings matched by schema H after the selection process, that is, at time t + 1 is given by

$$\mathcal{E}(H, t+1) = E(H,t) \{\frac{f(H,t)}{F(t)}\} \{pop\_size\} \tag{3}$$

We can simplify the above formula since $F(t)/pop\_size$ is the average fitness which can be written as $\overline{F}(t)$. Thus the formula becomes

$$\mathcal{E}(H, t+1) = \mathcal{E}(H,t) \{\frac{f(H,t)}{\overline{F}(t)}\} \tag{4}$$

This equation is often refered to as the *Reproductive Growth Equation.*

Now, consider the case where the schema H remains $\epsilon\%$ above the average fitness, then the number of strings matching H will be given by the equation

$$\mathcal{E}(H,t) = E(H,0)\ (1+\epsilon)^t \tag{5}$$

As we can see, this would mean that the number of strings matching the schema H is increasing exponentially in subsequent generations. In other words, through the process of selection, those schemata that has a high fitness compared with the average population fitness will increase in numbers as the generation evolves.

Next, during the process of crossover, a schema might be destroyed when segments of the chromosomes are swapped. The probability that any given schema would be lost during this process depends on the defining length of the schema and it is given by the formula

$$P_{cd}(H) = \frac{\delta(H)}{l-1} \tag{6}$$

Hence, the probability that a given schema would survive after crossover is given by

$$P_{cs}(H) = 1 - \frac{\delta(H)}{l-1} \tag{7}$$

However, not all of the given schema are destroyed in the process of crossover; some might have survived and new ones might even be formed from other schemata. As a result, the probability of schema survival is better than that expressed in equation 7 and it is closer to

$$P_{cs}(H) \geq 1 - \frac{\delta(H)}{l-1} \tag{8}$$

Therefore: after taking into account the effects of crossover, equation 4 becomes

$$\mathcal{E}(H,t+1) \geq \mathcal{E}(H,t)\ \{\frac{f(H,t)}{\overline{F}(t)}\}\ \{1 - p_c\frac{\delta(H)}{l-1}\} \tag{9}$$

where p_c is the rate of crossover.

Just as in crossover, mutation also influences schema survival since the operator might also change the component values of a potential schema. Given that the nmtation operator operates by changing the component bits of a chromosome with a certain propability p_m,

the proba'bility that any given schema would be destroyed depends on the number of fixed component bit in the schema, which is also the order of the schema. In this case, the higher the schema order, the higher the probability of destruction. Thus, the probability that a given schema would survive mutation is given by

$$P_{ms}(H) \approx 1 - p_m \; o(H) \tag{10}$$

where p_m is the rate of mutation. With that, equation (9) can be further improved to take into account the effects of mutation. This gives the final equation

$$\mathcal{E}(H, t+1) \geq \mathcal{E}(H, t) \; \{\frac{f(H,t)}{\overline{F}(t)}\} \; \{1 - p_c \frac{\delta(H)}{l-1} - p_m \; o(H)\} \tag{11}$$

Judging from the above equation, we can conclude that the schema that would survive and increase exponentially in subsequent generations is the one that is short, low-order, and has above average fitness performance. This is the essence of the ***Schema Theorem*** [7] which states that

Theorem 1 (Schema Theorem) ***Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.***

To summarize, the strength of a genetic algorithm lies in its ability to exploit information about the fitness of a large number of structural configurations without the computational burden of explicit calculation and storage. This allows a concentrated search of the solution space which contains solutions of above average fitness, culminating in the identification of the global optimal solution.

3 Optimal Solution List

In this section, we describe the optimal solution list as a means of solving the problem of locating the global optimal solution accurately in a noisy environment. In various practical applications, it is often impossible to evaluate the fitness or performance of each candidate solution accurately as it would be too computationally intensive. This is usually overcome

by approximating the performance using statistical sampling techniques. However, this would introduce noise into the performance measure evaluated, subsequently resulting in an inaccurate: optimal solution being indentified. This problem occurs when only one variable is used to record the optimal solution evaluated thus far.

We observe that the single variable does not constitute sufficient memory to maintian the best performing solution because the environment is noisy. The variable frequently records a better candidate solution which performed very well but would subsequently replace it with a less fit solution that appears to have a higher fitness value because of an added noise. As a result of this, the actual fitness of the candidate solution recorded by the single variable fluctuates up and down during the search process and depending on how fit the evaluated solutions *appear* to be, the final solution recorded may not be the global optimum but one that appears to be with the noise added. Therefore, we propose that an optimal solution list be maintained to record the best performing solutions found so far as opposed to a single variable in present implementations.

In this scheme, we maintain a small list to record a series of candidate solutions that have performed better than the rest with the noisy evaluations. During each iteration, when the fitness of each candidate solution is being evaluated, if the solution performs better than the worst performing solution in the optimal solution list, it would replace that solution in the list and this continues for the whole duration of the search. In this way, the list constantly maintains a set of candidate solutions which appear to have performed better than the rest. Consequently, even when the global optimum solution does not appear to be fitter than the rest, the probability that it will be identified through the optimal solution list will improve since it would still give a relatively good fitness measure. This is the motivation behind the optimal solution list.

In addition to the list, we also propose that the sampling rate of the evaluation be increased gradually along the process of the search. The intuition behirtd this is that in the beginning, when the population is still far away from the global optimum, it is not neccessary to spend effort on getting accurate evaluations, but as the process continues and the population converges towards the space around the global optimum, it would be more

worthwhile to allocate more resources to evaluate the candidate solutions accurately in order to identify the fitter candidate solutions with more confidence. This would help to improve the quality of the candidate solutions in the list. It is important to note that the accuracy of the fitness measure of both the population and the list should be increased at the same rate so that the effects of noise are equal in both during comparison.

In this scheme, when the sampling rate is increased, it is also necessary to decrease the population size to a certain minimum so that the overall computational effort spent in updating the list remains constant for the duration of the search. This is achieved by reducing the population size by one during each iteration while increasing the sampling rate each time the population size has reached $1/2$, $1/3$, $1/4$, ... of the original size. In this way, the total number of samplings for the entire search process would be equal to that of the simple genetic algorithm with the same initial population size and a sampling rate of one. This completes our description of the proposed solution.

4 Experiments

We have performed a series of experiments to test the performance of our proposed scheme. In these experiments, we use a simple genetic algorithm and a modified genetic algorithm with our proposed scheme to maximize the function

$$f(x,y) = 3\,(1-x)^2\,e^{-x^2-(y+1)^2} - 10\,(\frac{x}{2} - x^3 - y^4)\,e^{-x^2-y^2} - \frac{e^{-(x+1)^2-y^2}}{3} \qquad (12)$$

where $-3 \leq x, y \leq 3$. A gaussian noise with a mean of 0 and a variance of 8 is introduced into the system to test the effectiveness of the proposed scheme in a noisy environment.

First, we investigate the effects of noise on the average fitness of the population and the quality of the optimal solution evaluated at the end of the search. Next, we use the modified genetic algorithm with the optimal solution list (scheme A) to maximize the test function and the fittest solution in the list is recorded as the global optimum. List sizes of 5, 10, and 20 are tested to observe the effects of different list sizes. The population is maintained at the original size. Finally, we proceed to test the final version of our modified genetic algorithm with the proposed schemes of the optimal solution list and varying the population size and sampling rate dynamically (scheme B) on the same test function. In this case, we fix the list size at 10 and we vary the minimum size to which the population is reduced to. Four different minimum sizes of 16, 24, 32, and 50 are tested.

It is important to note that in the above experiments, even though we use the noisy test function for evaluating the fitness of each candidate solution in the evaluation and selection phase, we will use the test function without the gaussian noise to establish the true fitness measure of the candidate solutions in the list and in the single variable when plotting the graphs. This technique provides us with an accurate evaluation of the fittest candidate solution that can be obtained from the optimal solution list in the case of the modified genetic algorithm. There is no attempt to establish the fittest solution from the list based on noisy evaluations here because we assume that this solution can be easily established since the list is small and more intensive assessment of the fitness of those values can be easily accomplished.

5 Experimental Results

We now describe some experimental results obtained from our experiments. For each experiment, at least ten runs of the genetic algorithm were performed on the test function described in section 4. Note that all the experiments are run for an initial population size of 100 for 200 iterations. The initial sampling rate is set to 1 for all experiments.

Figure 1 shows the effects of noise on the selection of the fittest solution using a simple genetic algorithm. The fitness value of the best solution is evaluated for the whole duration of 200 iterations with a gaussian noise of mean 0 and variance 8. The result clearly shows the detrimental effects of noise on the performance of the genetic algorithm in terms of evaluating: the global optimal solution.

In the second set of experiments, we investigate the performance improvement as a result of introducing the optimal solution list to a conventional genetic algorithm. We evaluate the effects of varying sizes of the list. In the third set of experiments, we consider the effects of varying the population size from 100 to various minimum size, $MIN\_SIZE = 16$, 24, 32, 50, and increasing the sampling rate as explained in section 3 while maintaining the list size at 10. In the above experiments, the noise has a mean of 0 and a variance of 8.

Figures 2, 3, and 4 shows the results of the second set of experiments. It is clear from the stated figures that the proposed scheme of an optimal solution list does in fact improve the performance of the genetic algorithm. From figures 5, 6, 7, and 8, it is clearly shown that the additional scheme of the varying population size and sampling rate also helps to improve the performance. Our experimental results therefore demonstrate that the conventional genetic algorithm is sensitive to the effects of noise, whereas the proposal scheme: is highly robust and effective in locating the global optimal solution even in a noisy environment.

6 Conclusions

In this report, we proposed the use of an optimal solution list and the dynamic tuning of the sampling accuracy of the individual candidate solution and the population size in a genetic algorithm. The proposed scheme exhibits improved performance when compared to a conventional genetic algorithm. An additional advantage of our scheme is that it can be easily implemented in traditional genentic algorithm without much modifications. Our experimental results clearly demonstrate the effectiveness of our proposed scheme in locating the global optimum solution of a test function in the presence of noise.

We do not claim that this scheme will work for all kinds of problem as it has only been tested on a single type of test function. Further testing and evaluation of the scheme under various conditions are needed. Future research effort will include implementation of the proposed scheme in practical problems such as the optimization of a queuing system.

References

[1] L. Davis, ed., *Genetic algorithms and Simulated Annealing.* London: Pitman Press, 1987.

[2] L. Davis, ed., *Handbook of Genetic Algorithms.* 115 Fifth Avenue, New York, NewYork 10003: Van Nostrand Reinhold, 1991.

[3] K. A. De Jong, *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, 1975.

[4] J. M. Fitzpatrick and J. J. Grefenstette, "Genetic algorithms in noisy environment," *Machine Learning,* vol. 3, pp. 101–120, 1988.

[5] J. M. Fitzpatrick and J. J. Grefenstette, "Genetic search with approximate function evaluations," in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications,* (Pittsburg, PA), pp. 112–120, Lawrence Erlbaum, 1985.

[6] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with application to biology, control, and artificial intelligence.* Cambridge, Mass.: MIT Press, 1992.

[7] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs.* Berlin; New York: Springer-Verlag, 1992.

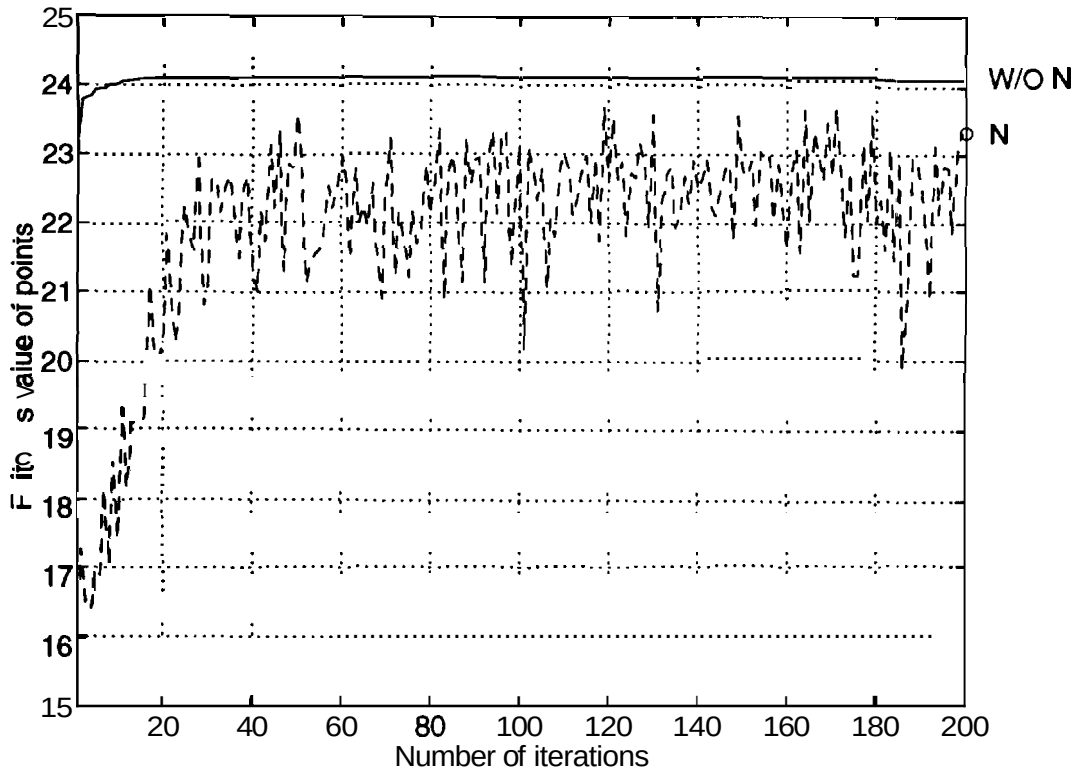


Figure 1: Effects of noise on a simple genetic alorithm

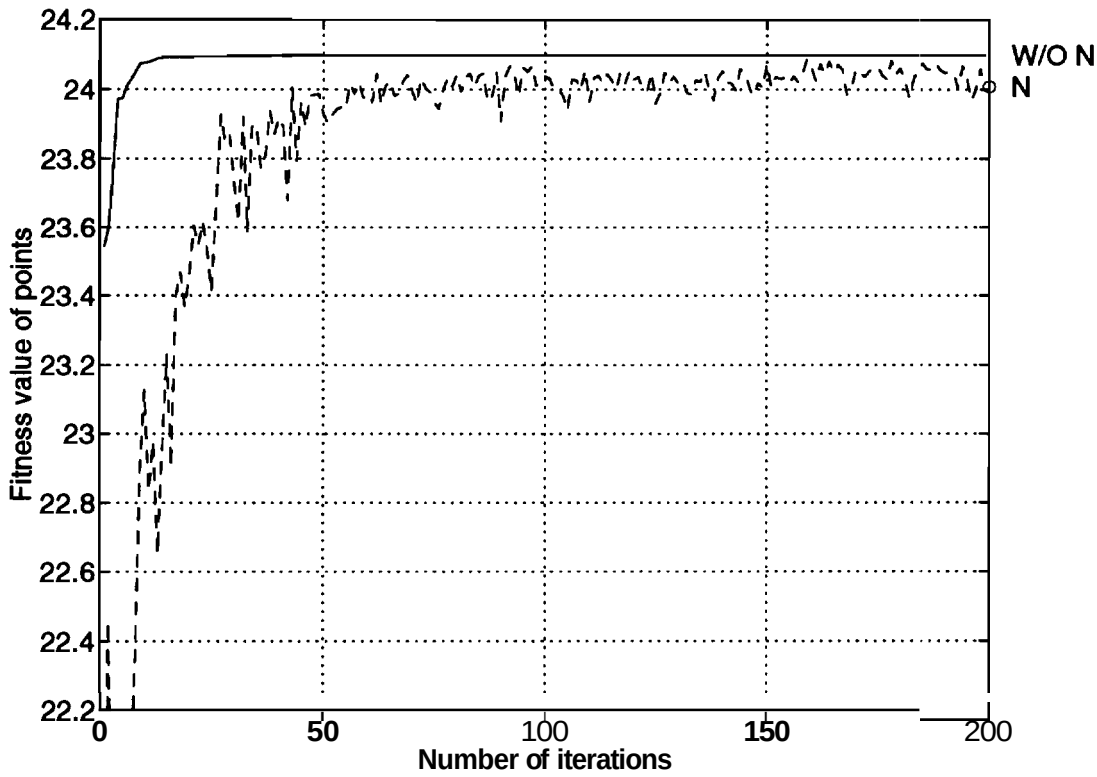


Figure 2: Scheme A: Minimum population size of 100 and list size of 5

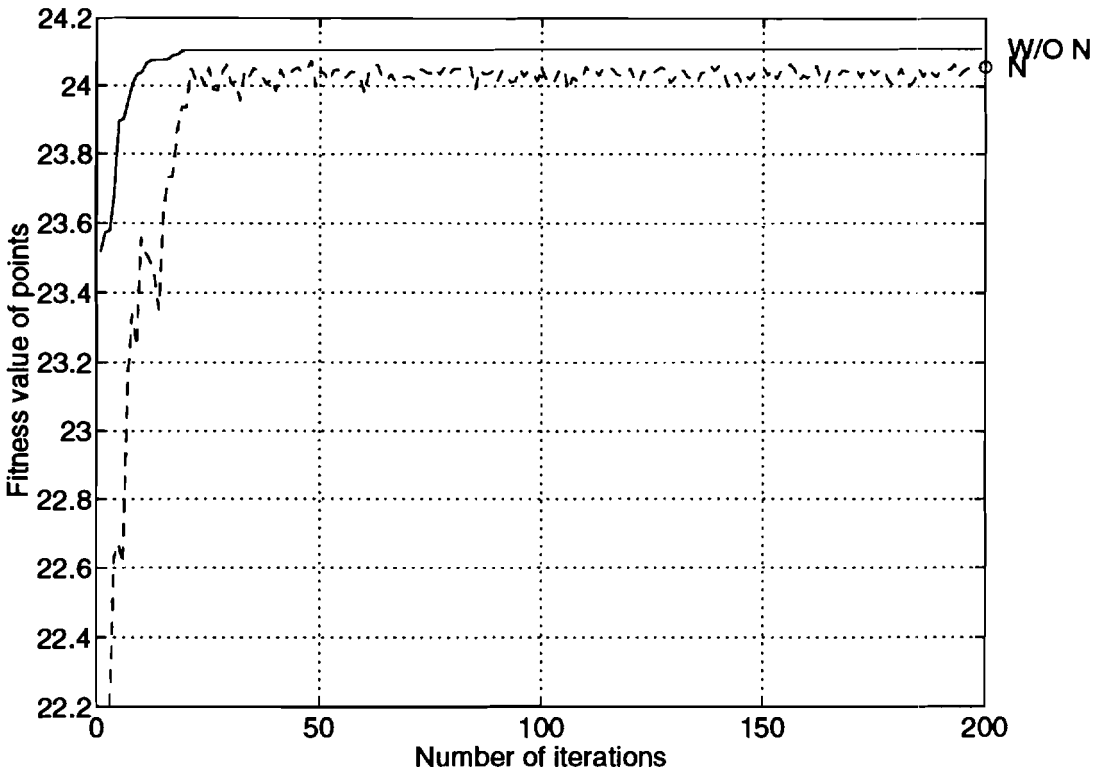
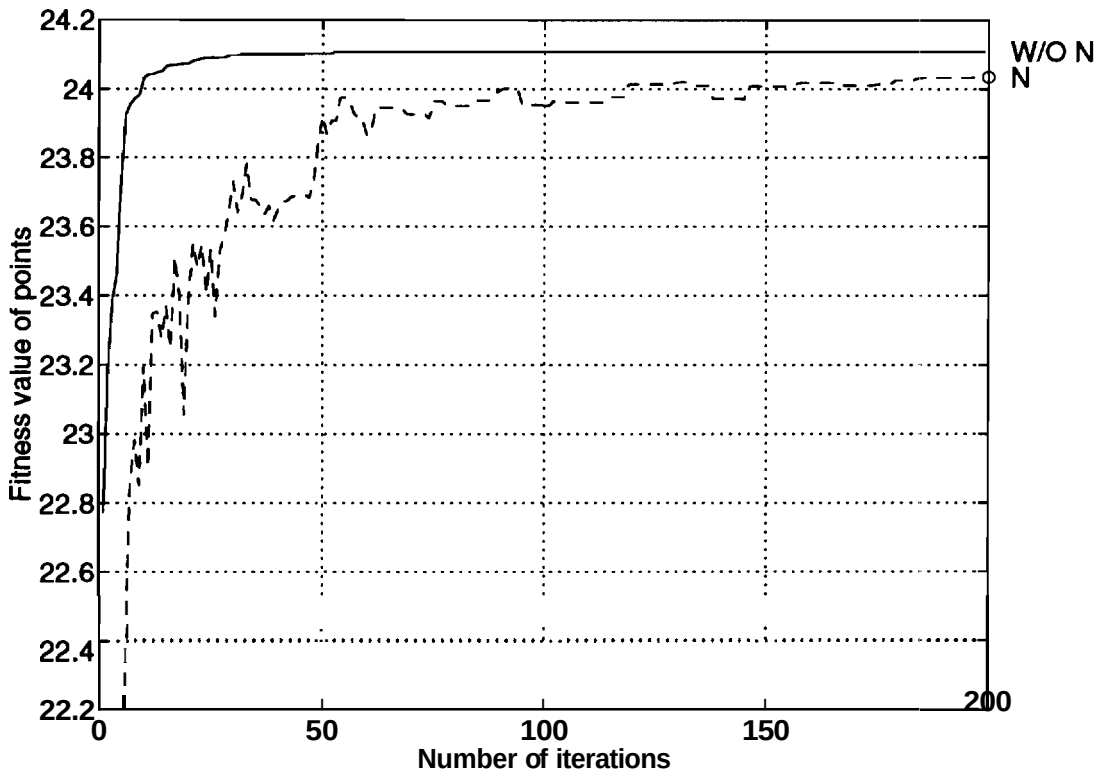


Figure 3: Scheme A: Minimum population size of 100 and list size of 10



:Figure 4: Scheme A: Minimum population size of 100 and list size of 20

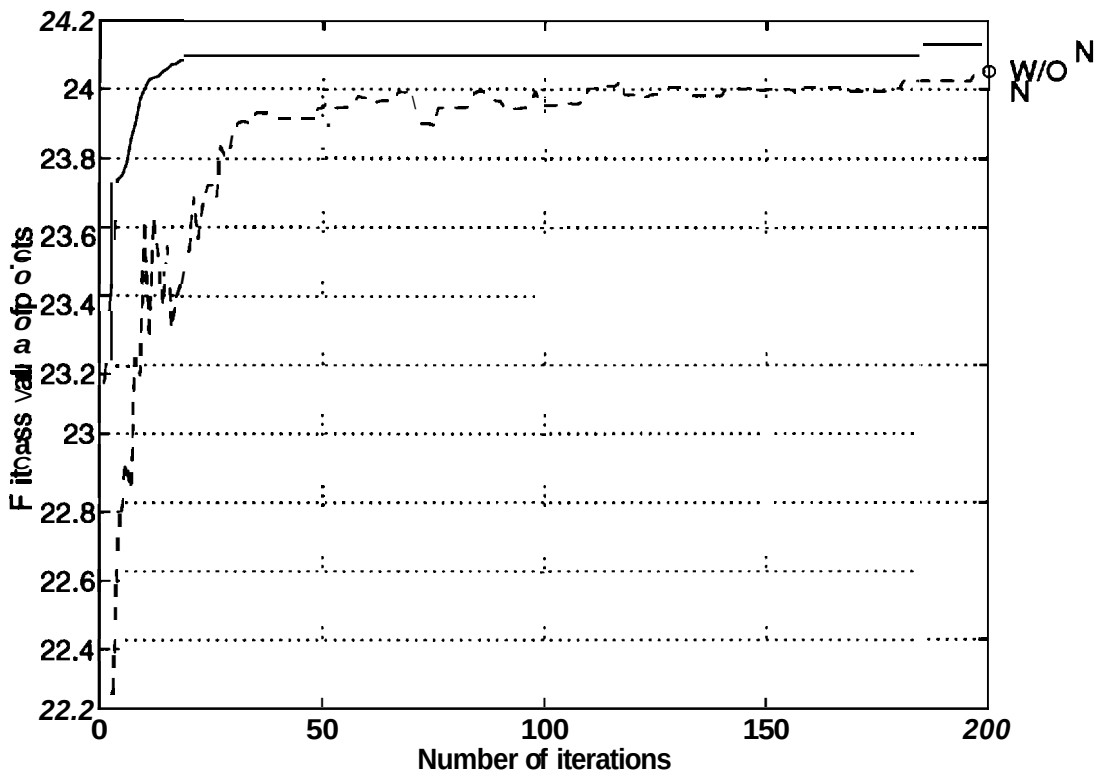


Figure 5: Scheme B: Minimum population size of 50 and list size of 10

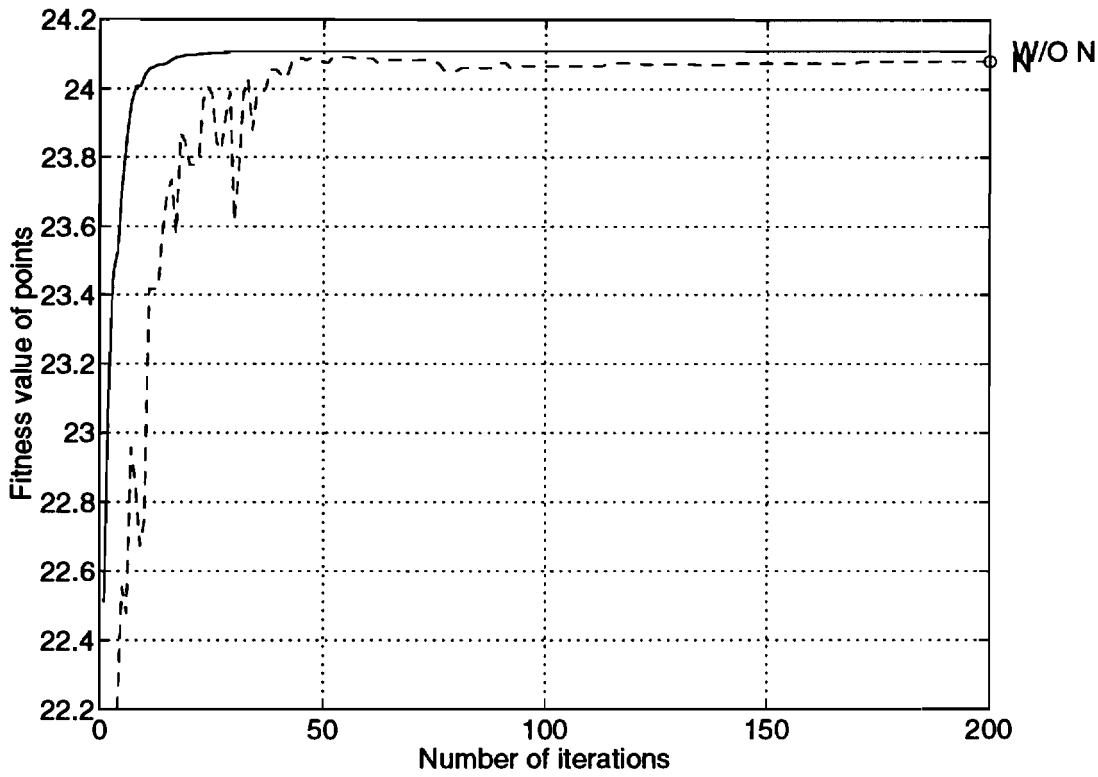


Figure 6: Scheme B: Minimum population size of 32 and list size of 10

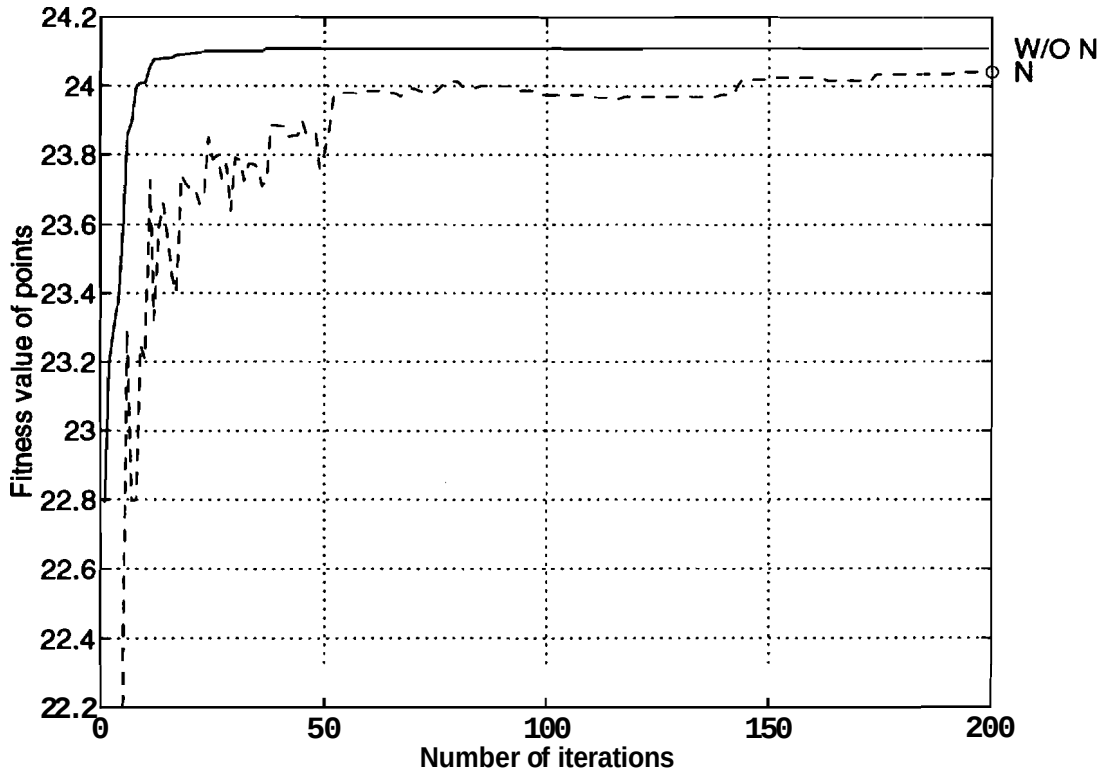


Figure 7: Scheme B: Minimum population size of 24 and list size of 10

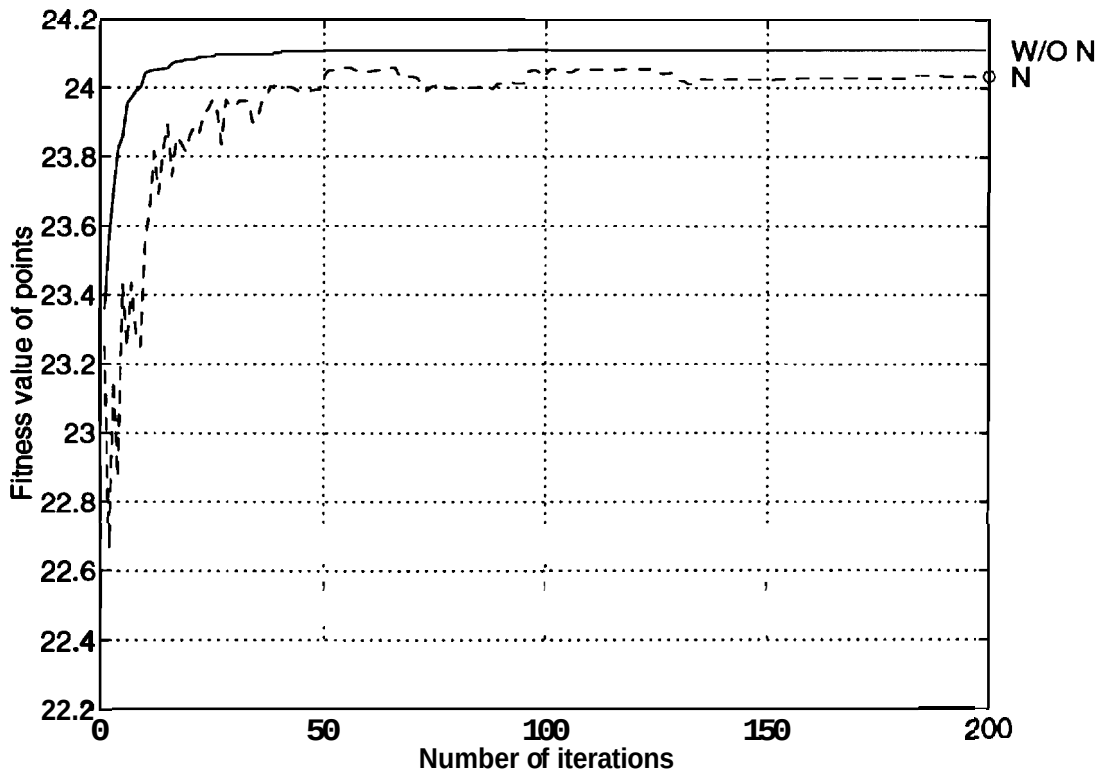


Figure 8: Scheme B: Minimum population size of 16 and list size of 10