Genetic Algorithms

Rafael E. Banchs

INTRODUCTION

This report discusses genetic algorithms, one class of global search techniques to be used in the inverse modeling of the time harmonic field electric logging problem. First, a brief description of its fundaments is presented. Then, a more precise description of the technique and the parameters involved in its implementation is discussed. Finally, an example is provided to illustrate the most important properties of genetic algorithms.

FUNDAMENTS AND PROPERTIES

As their name implies, genetic algorithms are based on the real processes of natural selection and survival of the fittest [1]. In biological populations, individuals of every particular species are constantly evolving and adapting to the surrounding environment. Such an adaptation process represents a biological analogy of the mathematical problem of maximizing an objective function, which in the case of genetics algorithms is referred as the fitness function.

In genetic algorithms, a set of artificial individuals (models) are used to define a population, and as in the case of biological systems, some genetic information is transmitted from generation to generation by a relatively simple set of combinatorial rules. While this evolution is taking place, the process of natural selection ensures that the fittest individuals are the ones with more probability of transmitting their genetic information. In this way, with the running of time, the individuals and the population are able to get more and more adapted to the environment; even if the environment itself changes with time.

The implementation of genetic algorithms requires the sub-codification of the model parameter space into a finite set of finite-length strings. The string, or group of strings, corresponding to a given model represents the genotype of the individual, while the model or individual itself represents the phenotype. According to the complexity of the problem, or the coding system, the strings in the genotype can be subdivided into chromosomes, which are composed by genes. Each of the single elements or characters of a string is called an allele and its position in the string is its locus. As it is explained later, in the particular examples considered along the present work, the terms gene and allele will be equivalent; however, under more sophisticated codification schemes a gene can be composed by more than one allele [2].

In the evolution of the algorithm, the individuals of a given population interchange their genotypes, according to their fitness values and some probabilistic transition rules, in order to produce a new generation. In this context, the objective function or fitness function provides an artificial selection criterion for giving to the best adapted individuals a higher chance to reproduce.

One of the most important properties of genetics algorithms is the robustness due to its parallel nature of search. As it is constantly searching from a population of models, instead of a single model, it has the capacity of performing a more extensive evaluation of the model space. In this way, as global search techniques in general, it does not present the problem of getting stuck in local minima. Also, despite the stochastic nature of the reproduction rules, it performs a guided search since the artificial selection criterion defined by the fitness function provides directivity behind the apparent randomness. So, there is no doubt that the most attractive feature of genetic algorithms is that, regardless they relative simplicity, they constitute a very powerful and robust searching technique.

Nevertheless, there is a simple but important difference between biological systems and artificial genetic algorithms, which actually represents the main disadvantage of this kind of searching techniques. That is the fact that, while nature has an eternity (or at least billions of years) for

playing the game of evolution, we certainly do not. Indeed, genetic algorithms are in general very slow and computationally expensive.

THE SIMPLE GENETIC ALGORITHM

The most simple genetic algorithm can be described as follows. First, it starts with a certain population of individuals selected at random from the model space. Then, an iterative procedure follows, in which each of its iterations is composed by three basic steps: selection, crossover and mutation. Finally, the algorithm is stopped when certain convergence criterion is achieved.

1.- Selection.

In this step, individuals are selected for crossover according to their relative fitness with respect to the others in the population. In this way, those individuals whose fitness values are above the population's average fitness will have more chance to reproduce than those whose fitness values are below the average. This selection step is implemented in practice by the creation of a mating pool. In the mating pool, certain amount of copies of each individual are placed according to its relative fitness. Figure 1 illustrates the mating pool corresponding to the presented population of five individuals.

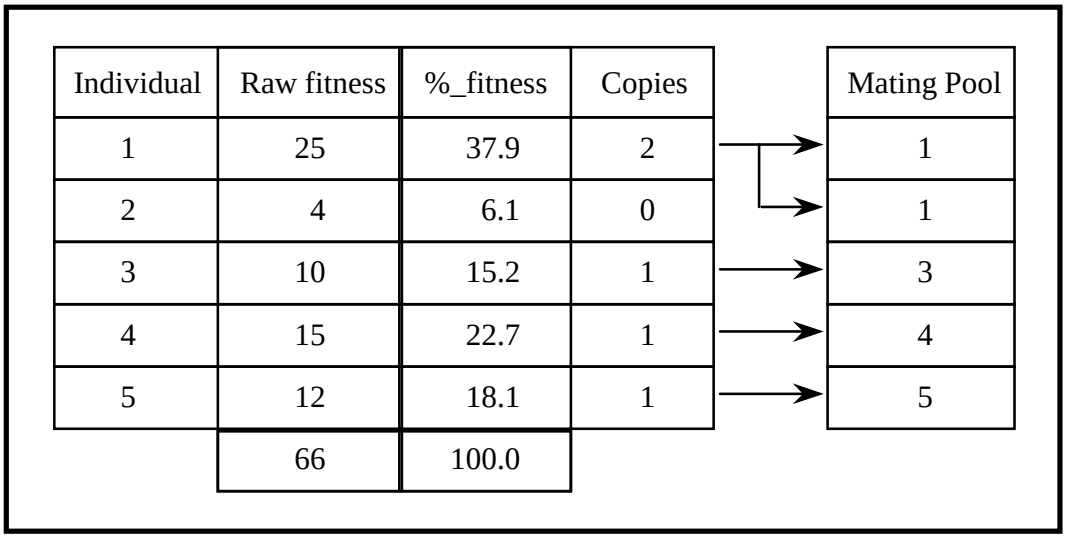

Figure 1: Selection by means of a mating pool.

The use of the mating pool resource permits the use of a uniformly distributed random generator during the step of crossover. However, in more complex implementations, it is always possible to combine selection and crossover in a single step by using a properly biased random generator.

2.- Crossover.

In this step, the genetic material of the individuals in the mating pool is recombined in order to produce a new generation. First, the individuals are picked in pairs from the mating pool. This is done uniformly at random. Then, for each pair, it is decided if crossover is going to be performed or not according to some probability p_c, which is called the crossover probability. If no crossover is required, both individuals are included just as they are in the new generation. On the other hand, if crossover is to be performed, their genotypes are used to generate a new pair of individuals. Figure 2 shows two different types of crossover schemes.

As it can be seen from Figure 2, in simple crossover, one locus is selected at random dividing the strings in two sections. Then, the alleles in one corresponding pair of sections are interchanged between the strings. In multiple crossover, more than one locus is selected at random dividing the strings in multiple sections. Then, corresponding non consecutive pairs of sections are swapped.

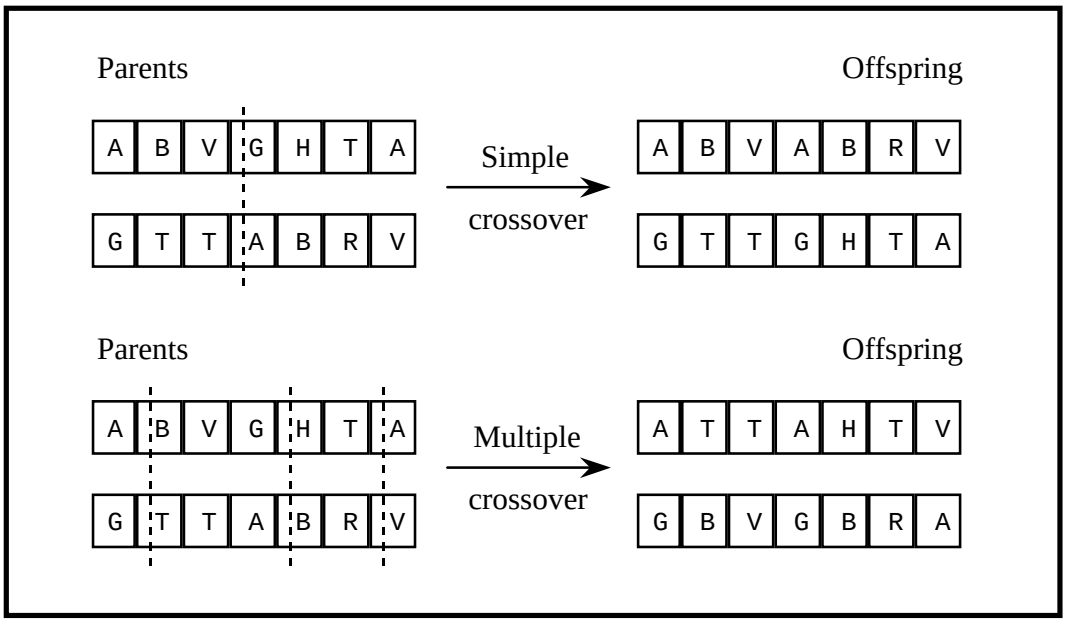

Figure 2: Simple and multiple crossover schemes.

There exist much more complex crossover and genetic coding schemes than the ones illustrated in Figure 2. Such is the case of diploid and triallelic forms of genotype that allows implementation of dominance and abeyance [2]. However, as it will be discussed later, the schemes presented in Figure 2 are suitable enough for the problem under consideration.

3.- Mutation.

In this step, some random alterations are performed to the genotypes of the individuals in the new generation. This is done according to certain probability p_m, the mutation probability. In general, p_m must be a very small value, such that the order of alterations would be around one in every thousand alleles. For every individual in the population, it is decided if a mutation must occur or not. Then, if it must occur, one locus is selected at random, and the value of its correspondent allele is altered.

In both, real life and genetic algorithms, mutation plays the very important function of avoiding irrecoverable losses of genetic material. If for example, the evolution of a population occurs in

such a way that at a given generation all the individuals have the same allele at certain locus. It is impossible, by means of the simple procedures described before (of course this is not the case for more sophisticated schemes), for such an allele to change ever again. In this way, mutation provides the possibility of recovering the genetic diversity for that particular allele.

A BRIEF MATHEMATICAL ANALYSIS

A more rigorous analysis of genetic algorithms can be performed by studying how the three basic operations of selection, crossover and mutation affect certain genetic structure or patterns called schemata [2]. Each pattern or schema represent a subset of models in the coded space.

Let us consider, for simplicity, a coding system that uses the binary alphabet {0,1}. Then the associated patterns to the given problem will be represented with the triadic alphabet {0, 1, X}; where the symbol X stands for a "don't care" character that can be either 0 or 1. In this way, in the case of strings of size 6, the pattern or schema 1XX0XX represents the subset of all the individuals of the population whose genotype is characterized by having a "1" in the 1st locus and a "0" in the 4th locus. Two important parameters of patterns must be defined. They are its order and its distance. The first is given by the number of non-X characters in it. The second is defined by the distance between the two outermost non-X characters in the string. Then, for the example provided above the order is 2 and the distance is 3.

Now, let us consider how different patterns or schemata are affected during one iteration of the genetic algorithm execution. During selection, as it was seen before, a particular individual $\bar{x}_i$ is picked with probability:

$$p(\bar{x}_i) = \frac{F(\bar{x}_i)}{\sum_{k=1}^{M} F(\bar{x}_k)} \qquad (1)$$

where $F(\bar{x}_n)$ is the fitness of the nth individual and M is the size of the population. Then, the expected amount of a particular pattern A in the resulting mating pool will be:

$$m_{A\_pool} = m_A \; M \frac{avf_A}{\sum_{k=1}^{M} F(\bar{x}_k)} = m_A \frac{avf_A}{avf} \tag{2}$$

where m_A represents the amount of patterns A present in the population, avf is the average fitness of the population, and avf_A is the average fitness among all individuals belonging to the subset defined by pattern A.

Next, during crossover, something interesting happens. For simplicity, let us only consider the case of simple crossover. Depending on the locus selected for cutting, a pattern may be broken or not. The probability of a given pattern A to 'survive' crossover is given by:

$$p_{sc}(A) \geq 1 - p_c \frac{d(A)}{N-1} \tag{3}$$

where p_c is the crossover probability, d(A) is the distance of pattern A and N is the total number of locus in the genotype (size of the string). However, notice that, depending on the genetic contents provided by the other parent, there exists a small possibility for pattern A to be recovered after crossover; for this reason (3) is defined by means of an inequality.

In a less extent than crossover, mutation may also destroy patterns. The probability of a given pattern A to 'survive' mutation is given by:

$$p_{sm}(A) = (1 - p_m)^{o(A)} \tag{4}$$

where p_m is the mutation probability and o(A) is the order of pattern A. In practice, the values of the mutation probability a much more smaller than 1; so, (4) can be approximated by:

$$p_{sm}(A) \approx 1 - p_m \, o(A) \tag{5}$$

Now, by combining the effects of selection, crossover and mutation, an estimate for the number of individuals associated to pattern A can be computed. It is given by the following expression:

$$m_A^+ \geq m_A \frac{avf_A}{avf} \left[1 - p_c \frac{d(A)}{N-1} - p_m \, o(A) \right] \tag{6}$$

Notice from (6) that the estimated number of copies of pattern A in the next generation is basically determined by the amount of copies in the present generation multiplied by two factors. The first factor is defined by a ratio of averages, which can be either greater or smaller than 1 depending on the average fitness of the individuals associated to pattern A. The second factor is defined by the algorithm probabilities and the particular characteristics of pattern A. This factor will be always smaller than the unit. However, it will be closer to 1 for patterns exhibiting small values of distance and order, while it will be closer to zero for patterns with large values of distance and order.

From the previous analysis, it can be concluded that the patterns with more chance to stay and multiply after successive generations, are those that satisfy both of the following conditions. First, they have to be compact; i.e. must exhibit small order and small distance. Second, they have to be highly fitted; i.e. must be associated to individuals presenting an over-average fitness value. Such a kind of patterns, that satisfy both conditions, are called “building blocks” [2].

In this way, such building blocks constitutes the essence of genetic algorithms’ performance. This is explained by the so called Building Block Hypothesis, which sustains that the convergence of genetic algorithms to the global optimum relays on the capacity of the adopted coding scheme to represent the best models in terms of building blocks. In other words, if the coding system is such that the best fitted individuals are composed by building blocks; then, according to (6), the algorithm will certainly find them and relatively fast. At this point, it is important to mention that in practice, the problem of designing an appropriate coding scheme can be as difficult as the optimization problem itself. However, this situation does not represent a actual trouble since, in most of the genetic algorithms’ applications, deceptive problems are not usually hard. This is due to their robustness and parallel-processing nature. In the most typical situations, inappropriate coding schemes will result in slower rates of convergence and more computational burden.

GENETIC ALGORITHMS AND THE THFEL PROBLEM

Among the most important aspects in the implementation of genetic algorithms for solving an specific problem are the selection of an appropriate coding scheme, the definition of the fitness function and the selection of the stopping criterion.

In the particular case of the time harmonic field electric logging problem (THFEL), the model is given by a set of logarithmic conductivities as it is described in [3]. The multiparameter nature of this problem then requires a coding scheme able to represent such a model space. One recommended scheme is the concatenated coding technique, which has been successfully used in multivariable optimization problems [2]. In this scheme, each parameter of the model is independently coded by using a conventional binary map of Nx bits. Then, all the coded parameters are concatenated together to form the string or genotype with a total length of M*Nx alleles; where M is the total number of zones in the formation.

In the implemented application, two different types of binary codes can be used. They are the unsigned binary code and the gray code. The range represented by the codes [0, 2^n] is then mapped into the interval $[-x_{max}, x_{max}]$ which represents the permissible values for the parameters of the logarithmic conductivity model $\bar{x}$. The resolution of the code is given by:

$$\Delta_x = \frac{x_{max}}{2^{(Nx-1)}} \tag{7}$$

Then, for a typical value of x_{max} = 9.21 (σ = 10^4 S/m), a representation of 10 bits per parameter will result in a resolution of about Δ_x = 0.0179 (Δσ = 0.0179 σ; about 2% of σ).

Another important aspect is the definition of the fitness function. For the same reasons presented in [3], we consider that the mean square error represents a good alternative to be used in the implementation of the method. However, as it was already mentioned, genetic algorithms constitute a maximization technique rather than a minimization one, but this problem can be easily solved by modifying the error function with an appropriate transformation.

Although the most simple way of doing so is by simply considering the negative of the error function, genetic algorithms require strictly positive fitness function. More suitable transformations are provided by the following definitions:

$$F(\bar{x}) = \frac{A}{\ln\left(k\,E(\bar{x}) + e\right)} \tag{8}$$

$$F(\bar{x}) = A\,\exp\left(-k\,E(\bar{x})\right) \tag{9}$$

where $F(\bar{x})$ represents the fitness function, A and k are some pre-defined scaling constants; and $E(\bar{x})$ is the mean square error defined by (1) in [3], or alternatively, its normalized version defined by (5) in [3].

The last consideration is related to the stopping criterion. Here, the most appropriate criterion will depend on the type of search under consideration. In the case of hybrid search, in which the objective is to provide the local search algorithm with a good starting model, a much more relaxed stopping criterion can be used. On the other hand, when stand alone global search is intended a more careful criterion must be considered in order to obtain the desired accuracy.

A typical stopping criterion in genetic algorithms is to wait until no significant variations are appreciated in the average fitness of the population. However, special care must be exercised with genetic algorithms. This is because they have a particular tendency, after running for long time, to sacrifice best fittest individuals in behalf of the average population. Although this problem can be corrected by the use of fitness scaling techniques, which are discussed in the next section; it is a good practice to keep always tracking of the best individual during the evolution of the algorithm.

IMPROVEMENTS TO THE BASIC ALGORITHM

In this section, additional features and variations of the simple genetic algorithm described before are described. The use of these options can help under some circumstances to improve the general performance of the inversion procedure.

1.- Fitness Scaling.

As it was already mentioned in the previous section, after running for long time, genetic algorithms have a particular tendency to sacrifice best fittest individuals in behalf of the average population. This happens because, after running for long time, the population is more homogeneous and the average fitness is very close to the best fitness value. This gives to the majority more chances to reproduce than the best individuals. In order to avoid this problem, the fitness value can be scaled in such a way that the best individual is guaranteed to have some small advantage over the average individual.

1.a.- Linear scaling.

One common scaling technique, is the linear scaling. It is done in the following way:

$$F_s(\bar{x}) = \frac{avf}{mxf - avf}\left[(k-1)F(\bar{x}) + mxf - k\ avf\right] \tag{10}$$

where $F_s(\bar{x})$ is the scaled fitness, $F(\bar{x})$ is the raw fitness, avf is the average raw fitness, mxf is the best raw fitness and k is the scaling parameter. The formula in (10) has been computed in such a way that the average fitness remains unchanged after the scaling is perform, and the maximum fitness becomes k times the average. A common value for k is 2.

Although, the linear scaling as defined in (10) solves the problem described before, it introduces the potential problem that after scaling, some of the smallest fitness can result into negative values. This new problem can be solved in three ways. One is by changing the linear scaling in such a way that the minimum fitness is scaled to zero. By doing so, the following is obtained:

$$F_s(\bar{x}) = \frac{avf}{avf - mnf}\left[F(\bar{x}) - mnf\right] \tag{11}$$

where mnf is the minimum raw fitness value.

Another alternative, is performing a simple truncation, in which all the negative scaled values are arbitrarily made zero. The last and more convenient alternative is the so called sigma truncation [4]. In this last procedure, the constant (avf-std) is subtracted from the raw fitness $F(\overline{x})$; where avf is the average raw fitness and std is the standard deviation of the raw fitness. After the subtraction, a normal linear scaling is completed by using (10); and finally, simple truncation is performed.

1.b.- Power Law Scaling.

In this scheme, the scaled fitness is computed as certain power of the raw fitness:

$$F_s(\overline{x}) = F(\overline{x})^k \qquad (12)$$

and although the value of k is generally problem-dependent, the values of 1.005 has been proved to work properly in some experimentation [5].

2.- Variable Population Size and Overlapped Populations.

In a variable population scheme, a variation of the population's size is permitted with probability p_{var}, at every generation. Once the variation occurrence have been determined according to p_{var}, one operation between increment or reduction is selected. Then, the size Δ of the variation is randomly computed between 1 and 10% of the total population size. In the case of increment, Δ new individuals are added to the existent population. In the case of decrement, Δ individuals are removed from the existent population. These added and removed individuals cam be selected at random or by following certain fitness related criterion.

In the case of overlapped populations [6], the size of mating pool is taken to be a certain fraction of the population size K. The amount of overlapping is defined by the overlapping parameter g. A value of g = 0, produces no overlapping and the current generation is totally replaced by its offspring. On the other hand, g = 1 would produce a total overlap, which means that no new generation is computed. Then, g must be defined in the interval [0, 1).

3.- Forced Diversity.

It is intended to avoid the collapse of an entire population into multiple copies of few individuals. This is common when the computational cost of a particular problem forces the use of relatively small populations. Diversity can be achieved by two simple ways described next.

3.a.- Incremented mutations.

Here, when the minimum and maximum fitness values are relatively close to each other, the mutation probability is multiplied by a large number. In this way, more genetic diversity is introduced when the population is becoming more homogeneous.

3.b.- Limited number of copies per generation.

In this scheme, the total number of copies that a particular individual can have is restricted. When the resulting genotype from a crossover is already in the population, a new crossover is performed or a new individual is generated at random.

Notice that the use of high diversity parameters in small populations turns the search more close to a random walk.

AN ILLUSTRATIVE EXAMPLE

The present section illustrates with a simple example the most important features of genetic algorithms. Two functions are considered in the example:

$$f_1(x) = \frac{1}{4}(x-3)^2 \tag{13}$$

$$f_2(x) = \frac{1}{2}\ \mathrm{Cos}\left(\frac{8\pi}{19}(x-1)\right) \tag{14}$$

where x represents an unidimensional model space, which is coded as an unsigned binary string of 10 bits mapped into the interval $-10.0 \leq x \leq 10.0$.

The inversion data set is defined as:

$$\bar{m}^T = [m_1\ m_2] = [f_1(1)\ \ f_2(1)] = [1\ \ 1/2] \tag{15}$$

And the fitness function is defined as by:

$$F(\bar{x}) = \exp\left\{-2\left[(f_1(x) - f_1(1))^2 + (f_2(x) - f_2(1))^2\right]\right\} \tag{16}$$

whose global maximum is located al x = 1, and presents a local maximum at x = 5.1.

Figure 3 illustrates the evolution of a fixed size population of 40 individuals. In this particular simulation no overlap was used, the crossover probability was 0.85 and the mutation probability was 0.001 with and incremental factor of 50. Linear fitness scaling with sigma truncation was used, and the individual number of copies per generation was restricted to 4.

Six plots corresponding to the original population and generations 15, 30, 45, 60 and 75 are presented. The plots illustrates the fitness function and the fitness values of each of the individuals in the population.

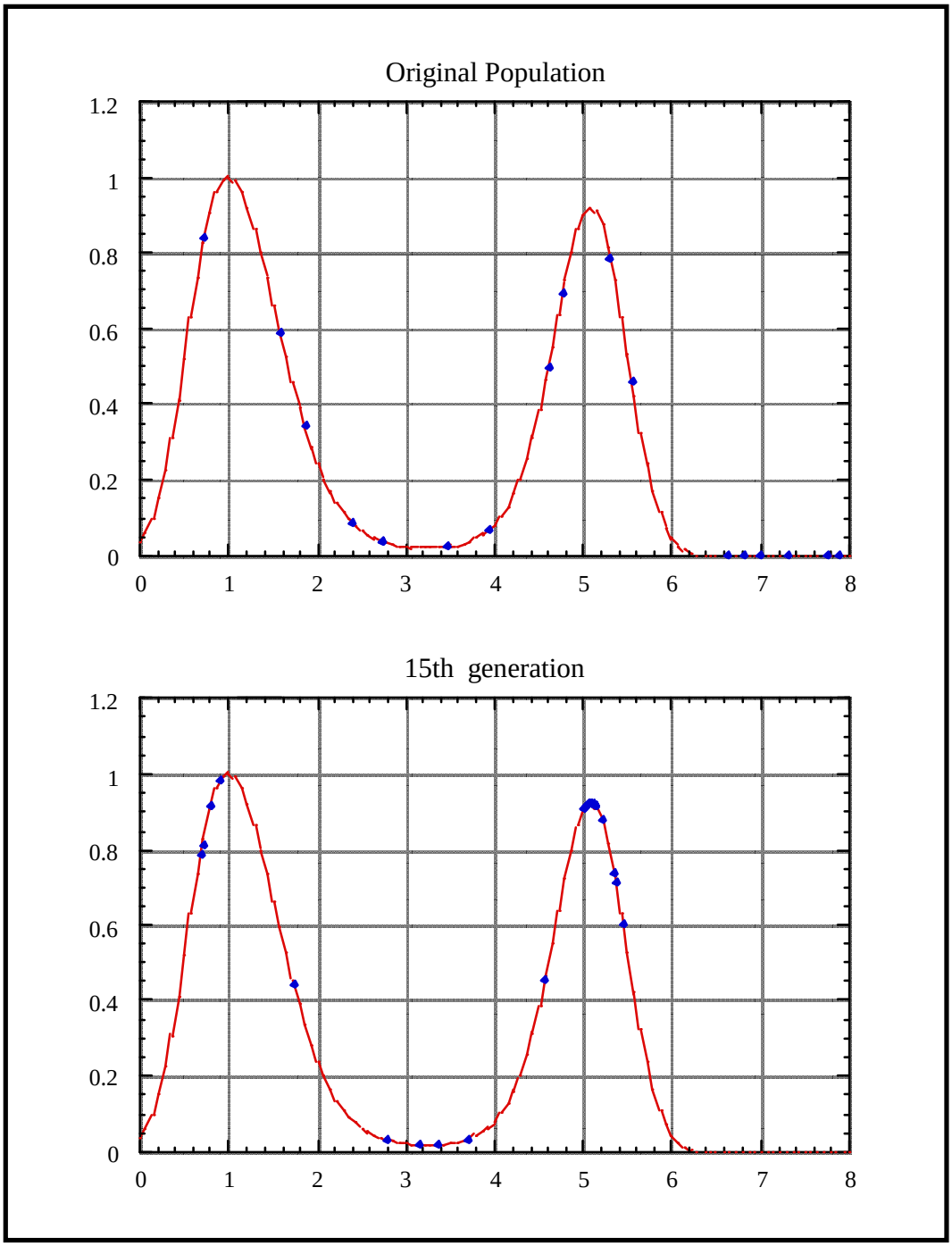

Figure 3: Evolution of a genetic algorithm.

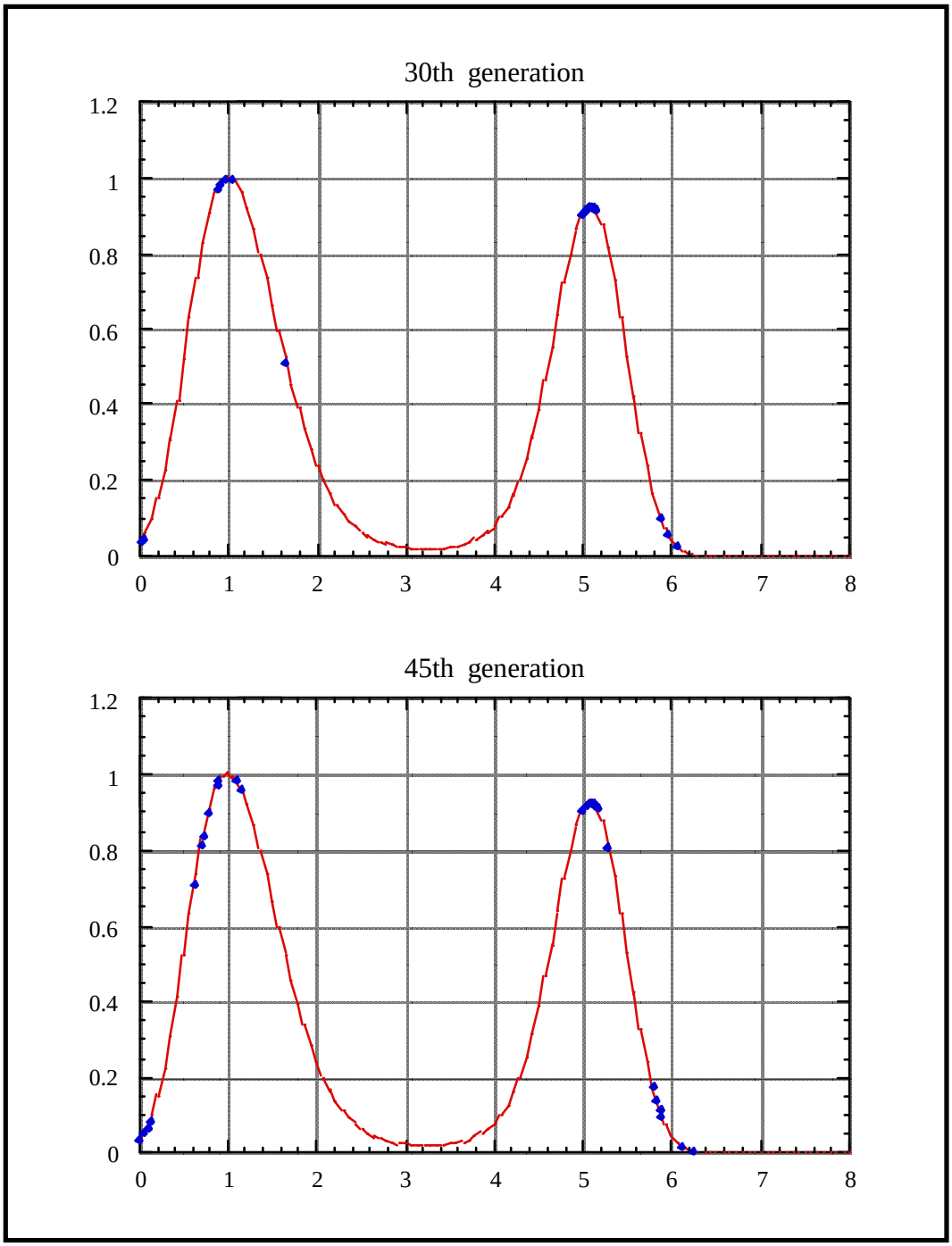

Figure 3: Evolution of a genetic algorithm. (Continuation)

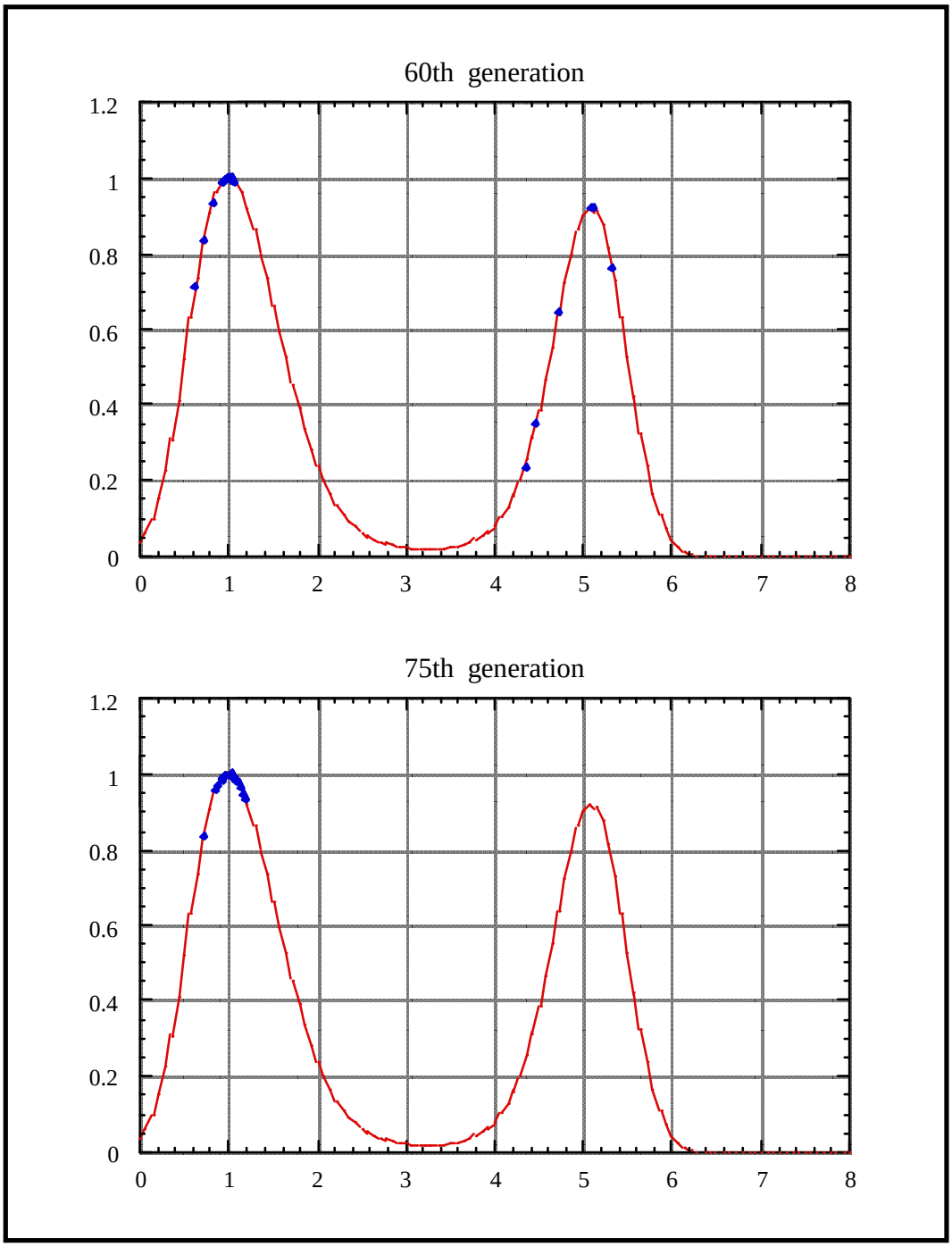
60th generation
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8
75th generation
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8

Figure 3: Evolution of a genetic algorithm. (Continuation)

From Figure 3, it can be observed how after starting from a uniformly distributed population the individuals tend to gather around the picks of the fitness function. This represents a very important property of the genetic algorithms. Due to their parallel nature of search, they have the ability of simultaneously locate different maxima over the fitness function surface. Notice how during the 30th generation the population is basically divided in two sub-populations, one around the global maximum and the other around the local maximum. From this point, some kind of confrontation between those sub-populations seems to begin. Indeed, notice how the diversity is notoriously increased at population 45. Finally, as it was expected, the algorithm converges to the correct solution.

This example also illustrates the computational intensity of this technique. For a population of 40 individuals evolving during 75 generations, a total of 3000 function evaluations where required. In practice, the total number of function evaluations can be reduced a little bit by generating a table of repeated individuals.

CONCLUSIONS

Genetic algorithms constitute an important family of global searching technique. Among its most important properties are their robustness and parallel processing capacity. However, as global search methods in general, their intensive computational requirements and very slow convergence characteristics are factors that reduce its potentiality in many cases.

In the particular case of the time harmonic field electric logging problem, in which function evaluations are computationally expensive, the feasibility of this technique depends basically on

the availability time. As in the case of simulated annealing, genetic algorithms are better suited for hybrid optimization schemes than for an stand alone global optimization.

REFERENCES

[1] Holland, J. (1975), Adaptation in Natural and Artificial Systems., Ann Arbor: The University of Michigan Press.

[2] Goldberg, D. (1989), Genetic Algorithms., Addison Wesley.

[3] Update Report #13: Gradient Methods.

[4] Forrest, S. (1985), Documentation for PRISONERS DILEMMA and NORMS Programs that Use the Genetic Algorithm., University of Michigan, Ann Arbor.

[5] Gillies, A. (1985), Machine Learning Procedures for Generating Image Domain Feature Detectors., University of Michigan, Ann Arbor.

[6] De Jong, K. (1975), An Analysis of the Behavior of a Class of Genetic Adaptive Systems, University of Michigan, Ann Arbor.