Pergamon

0098-3004(94)E0031-N

Computers & Geosciences Vol. 20, No. 7/8, pp. 1229–1236, 1994

0098-3004/94 $7.00 + 0.00

GENETIC ALGORITHMS: A POWERFUL TOOL FOR LARGE-SCALE NONLINEAR OPTIMIZATION PROBLEMS

KERRY GALLAGHER[1]* and MALCOLM SAMBRIDGE[2]†

[1]School of Geological Sciences, Kingston University, Penrhyn Road, Kingston-upon-Thames KT1 2EE, U.K. and [2]Research School of Earth Sciences, Institute of Advanced Studies, Australian National University, Canberra, ACT 2601, Australia

(Received 15 August 1992; revised 18 January 1994)

Abstract—Genetic algorithms represent an efficient global method for nonlinear optimization problems, that are encountered in the earth sciences. They share the favorable characteristics of random Monte Carlo over local optimization methods in that they do not require linearizing assumptions nor the calculation of partial derivatives, are independent of the misfit criterion, and avoid numerical instabilities associated with matrix inversion. The additional advantages over conventional methods such as iterative least squares is that the sampling is global, rather than local, thereby reducing the tendency to become entrapped in local minima and avoiding a dependency on an assumed starting model. In contrast to random Monte Carlo, however, they also share a desirable characteristic of the local methods in that they assimilate and take advantage of information collected during the sampling of the model space, resulting in an extremely efficient and robust optimization technique. This paper describes the basic genetic algorithm, briefly highlights some recent applications in the earth sciences and concludes that, in this field, the methodology should have many applications.

Key Words: Genetic algorithms, Monte Carlo simulation, Nonlinear optimization, Inversion.

INTRODUCTION

In the earth sciences many important problems that require data fitting and parameter estimation are nonlinear and can involve many unknown parameters. Consequently, the application of analytical inversion or optimization techniques may be restricted. In practice, the analytical methods, such as least squares or steepest descent, are local in nature and rely on a linearized form of the problem in question, adopting an iterative procedure using partial derivatives to improve on some initial model. This approach can lead to a dependence on the starting model and is prone to entrapment in local minima. Moreover, the calculation of derivatives can be difficult and further add to instability if numerical approximations are used. In contrast to these local methods, global techniques do not rely on partial derivatives, are independent of the form of the data misfit criterion, and are computationally robust, for example by avoiding matrix inversion. Global methods may use random processes to sample a wider span of the model space and uniform Monte Carlo sampling probably is the most well known of these techniques. In this situation, randomly generated models are assessed in terms of their data-fitting quality and the process may be stopped after a certain number of models or continued until a satisfactory data fit is achieved.

The nature of this random searching for near-optimal solutions involves a large degree of potentially wasteful computation through sampling unfavorable regions of parameter space. The process of randomly exploring relatively large regions of parameter space is in stark contrast to local methods, which exploit the information gained through the sampling of only a few models and their partial derivatives to reduce the data misfit. Therefore, a tradeoff exists between robust exploration of the parameter space and efficient exploitation of the information provided from this sampling. A general, nonproblem-specific optimization method ideally should combine both of these desirable characteristics. A new class of methods known as genetic algorithms achieve this end through novel model representation and manipulations.

Here, we discuss the basics of genetic algorithms, drawing attention to some of the more accessible literature and recent applications in the earth sciences. As this is a rapidly evolving field, this paper is not meant to be a comprehensive tutorial nor exhaustive review. Rather, the aim is to present a brief introduction to genetic algorithms, with an emphasis on their application to optimization problems. A more extensive discussion of the background, applications, and variations in the methodology, of which

*†Formerly at: *Department of Geological Sciences, University College London and †Institute of Theoretical Geophysics, Cambridge University, U.K.

there are many, are contained in the references cited and additional references therein.

Genetic algorithms were developed originally in the field of artificial intelligence by John Holland more than 20 years ago (Holland 1975, 1992a, 1992b), but even in this field, it is only in the last 5 years or so that the methodology has been applied more generally. Examples of their use in a variety of applications and other relevant issues are given in Grefenstette (1985, 1987), Shaffer (1989), Belew and Booker (1991), Goldberg (1989), Rawlins (1991), Davis (1990, 1991), Riolo (1992), Forrest (1993), and Whiteley (1993).

The methodology only recently has attracted attention in the earth sciences literature. To date, the applications generally have been concentrated in geophysics, and in particular seismology. Stoffa and Sen (1991) and Sen and Stoffa (1992) have used the approach for fitting synthetic body waveforms for 1-D depth-dependent seismic profiles; Gallagher, Sambridge, and Drijkoningen (1991) and Sambridge and Drijkoningen (1992) applied genetic algorithms to the inversion of real marine refraction data to constrain crustal structure; Wilson and Vasudevan (1991), studied their use in residual statics estimation; Scales, Smith and Fischer (1992), and Smith, Scales, and Fischer (1992), are concerned with reflection waveforms, whereas Kennett and Sambridge (1992) and Sambridge and Gallagher (1993) have proposed their use in earthquake hypocenter location. Other seismological applications include Jin and Madriaga (1993) on inversion for the background velocity field in seismic reflection and Nolte and Frazer (1994) in vertical seismic profiling. New areas where the methods may have useful applications in seismology include estimation of near-source seismic structure (Zhou, Tajima, and Stoffa, 1993), phase delay estimation (Winchester, Creager, and McSweeney, 1993), and reflection/refraction modeling (Kappus and Harding, 1993). More recently, the methodology has been used to infer thermal histories from apatite fission track analysis (Gallagher, Hawkesworth, and Mantovani, 1994a) and model the evolution of regional topography at rift margins (Gallagher, Hawkesworth, and Mantovani, 1994b). As awareness of genetic algorithms grows there surely will be many more and varied applications to earth science problems.

EVOLUTION AND THE PHILOSOPHY OF GENETIC ALGORITHMS

As related by Davis (1991), the features of evolution intrigued John Holland about 20 years ago. Specifically, Holland believed that computer algorithms simulating evolution could be developed to evolve solutions to complex problems (see Holland, 1992b). It generally is accepted that evolution occurs by natural selection so that, within a given environment, more successful organisms survive and propagate, whereas the less well adapted decline. We could regard evolution as an effectively self-optimizing process in that the evolving system does not know a priori what constitutes a successful organism. Furthermore, the current population of organisms has no memory of what has gone before. The complex mechanisms of evolution are not particularly well understood but some general characteristics emerge.

Firstly, evolutionary changes occur at a molecular (or chromosome) level and it is the combination of these changes that leads to the macroscale evolutionary characteristics with which most of us are familiar. For example, certain genes determine the color of hair and eyes or susceptibility to disease. Thus, we can regard the chromosome structure (or genotype) as a particular encoding of the characteristics of the macroorganism. The individual chromosomes, however, have no knowledge of what effect changes to their structure have on the macroscale yet the survival of a particular chromosome structure depends on how well the macroorganism performs in its environment.

The second relevant characteristic of evolution is that changes to chromosomes and genotypes occur during reproduction and these changes are facilitated through the relatively simple processes of crossover and mutation. Reproduction generates offspring and crossover allows the chromosome structure of the parent organisms to be modified, by the exchange and recombination of parts of each parent structure. The crossover process allows offspring to have a combination of the parents' characteristics and the offspring also may develop some different features, depending on how chromosome structures are recombined. Mutation is a random process which also provides the opportunity to introduce new characteristics unrelated to the parents. In the general scheme of evolution, mutation generally is regarded as secondary to crossover. In part, this is because mutation occurs relatively infrequently but, more importantly, it is a less efficient optimizing process because it fails to exploit the information contained in the parent structures which contribute to successful organisms.

The basic idea in using genetic algorithms as an optimization method is to represent a population of possible solutions, or models, in a chromosome-type encoding, and manipulate these encoded models through simulated reproduction, crossover, and mutation. Those sample models with more characteristics in common with the correct solution (i.e. the most successful possible model) would tend to survive during the evolutionary process, whereas the less successful models would die off, in a manner analogous to the survival of the fittest in nature.

To achieve this model it is necessary to develop a suitable encoding and a method of evaluating the success of an individual model. Significantly, however, the algorithm does not need to know the details of the problem it is trying to solve. In terms of optimization problems then, the advantages of

genetic algorithms are that the general implementation is independent of the nature of both the forward problem and the form of objective function in that we avoid the need to calculate partial derivatives or perform matrix inversion. However, the actual performance of a particular genetic algorithm can be problem dependent, and the modifications required to handle this aspect is a major area of current research in the field.

THE BASIC GENETIC ALGORITHM

Genetic algorithms use stochastic processes to produce an initial population of models, in similar fashion to random Monte Carlo simulation, and simple manipulations or operators are applied to the model population. The result of applying the operators to a model population is to produce a new population of models, constituting an iteration (Fig. 1). This process is repeated a number of times until a suitable model or group of models evolves. In genetic algorithm terminology, the initial population is referred to as parents, the new population as offspring, and each iteration represents a successive generation. Before we can invoke this iterative approach we need to consider the model representation and evaluation.

Model representation

A key aspect of genetic algorithms is the representation of complex models by simple encoding. The encoding initially considered by Holland (1975), and probably the simplest conceptually, is the representation of a model by binary digits or bit strings. There are many other types of encoding, which incorporate higher cardinality than binary (e.g. real numbers) or rely on ordering rather than direct representation of parameter values. As stated by both Goldberg (1989) and Davis (1991), the best encoding is problem-specific and may require some experimentation and modification of the crossover and mutation operators. Indeed, Davis (1991) states that none of his nine real-world applications of genetic algorithms have incorporated ultimately the binary encoding. However, this form of encoding is widely used, being easy to code and manipulate, and therefore may be the starting point for a given problem. Once some insight is gained into the behavior of the binary-coded genetic algorithm, it may become more obvious that an alternative coding is more suitable for a particular problem. In this overview, we concentrate our attention on the binary or bit-string coding, but emphasize that alternatives are available and can be more powerful.

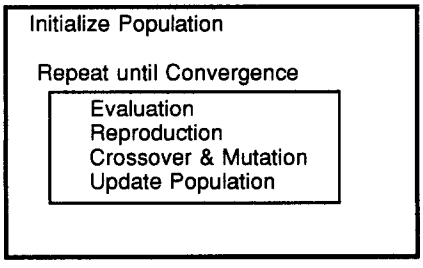

Figure 1. General scheme for genetic algorithm.

When we consider a binary coding for a series of models, the patterns of 1s and 0s in the individual binary strings then represent characteristics of the corresponding model. The aim of the genetic algorithm is to combine and propagate the characteristics of good models to produce even better models. We do not want to enter into the mathematical analysis of how genetic algorithms using the binary encoding achieve this and accessible discussions of this aspect are given in Goldberg (1989) and Forrest (1993). However, the important results of such analyses are that effectively many models are sampled simultaneously because they have similar bit patterns and the patterns which contribute to the better models are sampled preferentially as the genetic algorithm progresses. The former characteristic is known as implicit parallelism.

Model evaluation

The other important aspect is how to specify the model evaluation criterion. In the context of genetic algorithms, this criterion is referred to as the fitness, and then we invoke the genetic algorithm, we are looking for the fittest (or at least the fitter) models, in the sense of the survival of the fittest. Not only do we need to be able to say whether one model is better than another, but also how much better. For many problems of interest in earth science, this is reasonably straightforward as we are concerned with observed data, and may be interested in determining a set of parameters (for a physical model) that provide a good prediction of the observed data.

In conventional optimization we may seek to minimize data misfit (the difference between the observed data and predicted values), whereas the genetic algorithm wants to maximize the fitness. The transformation from one to the other is trivial, and can be approached in the same way as maximum likelihood estimation, or simply by changing the sign of the misfit value.

The problem of deciding how much better a particular model is requires a little more thought. As it progresses, the genetic algorithm is designed to retain the better models preferentially through stochastic sampling processes so that the fitter models are more likely to survive and procreate. In effect, that probability of a given model being sampled and selected for the next stage of the genetic algorithm depends on its fitness value. However, we can envisage sampling a reasonably good model (i.e. relatively high fitness)

early on in the progress of the algorithm. Although this model may be far from the best possible, it may be significantly better than the others sampled to date. In this situation, the algorithm will tend to stall by converging on this model. This characteristic is referred to as premature convergence. Another problem can arise when all models in the current population yield similar, but not necessarily high, fitness values. As a result of the stochastic sampling, the worst models in this mediocre population may be sampled as often as the best. However, both of these problems usually can be avoided using a transformation, or scaling, of the fitness function (see Goldberg, 1989).

A binary-coded genetic algorithm

It is useful to discuss an example of a genetic algorithm and for this we retain the binary encoding as outlined. We shall return to some of the variations subsequently. The general problem that we consider is the following: a set of unknowns, x_i, $i = 1, M$, is represented by the model vector $\mathbf{m}$ and a related objective function is given as $\phi(\mathbf{m})$. In general, $\phi(\mathbf{m})$ will represent a data misfit, reflecting the mismatch between a set of observed data and values predicted for a given $\mathbf{m}$, but additional desirable model properties such as smoothness constraints may be incorporated into the objective function. In practice, we are interested in determining the models that can predict the observed data adequately, that is less than a specified limit of $\phi(\mathbf{m})$ determined from some knowledge of errors in the data. We have noted the relationship of misfit to fitness earlier and for the remainder of this paper, we shall equate $\phi(\mathbf{m})$ to the fitness function and the objective is to maximize this function.

The individual model parameters each have a specified range so that $x_i^{\min} \leqslant x_i \leqslant x_i^{\max}$ and the minimum allowable variation in each model parameter is determined by specifying a discretization interval Δx_i so that the N_i possible values of x_i are given as $x_i^{\min} + j \cdot \Delta x_i$ for $j = 0, N_i - 1$. As a consequence of the binary-string coding that will be used subsequently, we require each N_i to be an integral power of 2 and then the value of Δx_i is given as $(x_i^{\max} - x_i^{\min})/(N_i - 1)$ and the total number of models in the discrete model space is

$$\prod_{i=1}^{M} N_i.$$

Our basic genetic algorithm proceeds as follows: Q randomly selected models (with M model parameters, x_i, $i = 1, M$) form the initial population. Each model is coded into a binary string, with the length of the binary string, L, determined as

$$\frac{1}{\log_e(2)} \sum_{i=1}^{M} \log_e(N_i).$$

The bit string of an individual parameter, x_i, is the binary number equivalent of the integer K_i, where

$$K_i = (N_i - 1) \cdot \frac{(x_i - x_i^{\min})}{(x_i^{\max} - x_i^{\min})}.$$

For example, suppose that we were interested in finding the x–y coordinates of the maximum of some 2-D function with bounds on both the x and y coordinates of 0 and 31. Selecting $N_x = N_y = 32$, then $\Delta x = \Delta y = 1$ and the total string length is equal 10 bits. The point (9, 23) then could be represented by the string (0, 1, 0, 0, 1, 1, 0, 1, 1, 1), where the first 5 bits equal 9 and the second 5 bits equal 23.

Having defined a population of Q models and calculated the value of the fitness function $\phi(\mathbf{m})$ for each model, an iteration of our genetic algorithm proceeds in three stages, corresponding to the operations as mentioned and these now will be described in turn.

(i) **Reproduction.** This stage selects an interim population of Q models via some stochastic selection process. As the aim is to propagate the better or fitter models, those with higher values of the fitness function (or equivalently lower values of data misfit) should have a higher probability of proceeding to the next model generation. The intuitive measure of this probability then is the value of the fitness function itself or, if necessary, a scaled fitness value, as mention earlier. One of the simplest procedures to select models to pass into the interim population is known as tournament selection. This method is based on relative rank, rather than the absolute value of fitness. Two models are selected at random from the current population of Q models and a random number between 0 and 1 is generated. If the random number is less than the specified value of tournament success probability, P_{TS}, then the higher fitness model proceeds into the interim population. Otherwise, the lower fitness model is selected. Both models are replaced in the current population, and the process is repeated until the interim population also has Q models.

(ii) **Crossover.** Having selected an interim, or parent, population, we want to produce an offspring population and this is achieved by randomly pairing off members of the parent population. Once all parents have been paired off to form $Q/2$ pairs, each pair is selected progressively and a random number between 0 and 1 is generated. This number is compared to a specified crossover probability, P_c. If the random number is greater than P_c, then the two parents pass into the next generation unchanged. If not, then the two parents are crossed over. The crossover operation involves the random selection of a position along the bit-string representation and exchanging that part to the right of the crossover position in the bit-string of one parent with the equivalent part in the other parent. For example, consider a pair of parents given by our previous bit string of (0, 1, 0, 0, 1, 1, 0, 1, 1, 1) and another

Table 1. Progress of simple genetic algorithm to determine maximum of $\phi(x) = x^2$ for $0 \leqslant x \leqslant 127$, with $Q = 4$, $P_{TC} = 0.7$, $P_c = 1.0$, and $P_M = 0.0$

Generation 1							
i	x_i	Bit-string	$\phi(x)$	Parents	Crossover	Offspring	x_i
1	15	0001111	225	1	000-1111	0000101	5
2	70	1000110	4900	3	110-0101	1101111	111
3	101	1100101	10201	2	1-000110	1111000	120
4	56	0111000	3136	4	0-111000	0000110	6
ϕ_{av}			4616				6696
Generation 2							
i	x_i	Bit-string	$\phi(x)$	Parents	Crossover	Offspring	x_i
1	5	0000101	25	2	110111-1	1101110	110
2	111	1101111	12321	3	111100-0	1111001	121
3	120	1111000	14400	2	1101-111	1101000	68
4	6	0000110	36	3	1111-000	1111111	127
ϕ_{avg}			6696				11874

string, (1, 1, 1, 1, 0, 0, 1, 1, 0, 0), representing the point (30, 12). If the crossover position is selected to be 4, then, counting from the left end of the string, the strings after crossover are (1, 1, 1, 1, 1, 1, 0, 1, 1, 1) and (0, 1, 0, 0, 0, 0, 0,1, 1, 0, 0), giving the pairs (31, 23) and (8, 12).

(iii) Mutation. In order to maintain some random character in the search process, a mutation probability, P_M, is specified. All bit positions are tested for mutation by generating a random number and comparing it to P_M. If it is less than P_M, then the bit is flipped in parity, that is 1 is changed to 0 or vice versa. Otherwise, the bit is unchanged. In practice, mutation is regarded as providing a restricted or local search capability. Thus, if $P_M = 1$, then all bit locations may undergo mutation, and the algorithm will perform similarly to random Monte Carlo. However, if $P_M = 1/L$, where L is the number of bits in a string, then, on average, one bit per string will undergo mutation, and randomization is greatly reduced. Mutation is important as it introduces diversity in the model population, which reproduction and crossover cannot achieve.

A simple step-by-step numerical example is the best way to illustrate the nature of the implementation. Let us consider the trivial problem of determining maximum value of x^2 for $0 \leqslant x \leqslant 127$ and so we can equate $\phi(\mathbf{m})$ to x^2. This is illustrated in Table 1 for $Q = 4$, $P_{TS} = 0.70$, $P_c = 1.0$, and $P_M = 0.0$, so that crossover always is performed and mutation never is performed. In this example, all four initial models pass through to the first interim parent population. This indicates that the fitness function has no influence on this particular reproduction step, an artifact of the small and finite sample taken. However, the two poorer models drop out in the second parent population as expected. After the second iteration, we actually reach the optimal solution, although this is fortuitous.

Even in this simple example, a feature to note is that the average fitness of each generation increases, indicating that the model population as a whole is improving through the exchange of information effected by crossover. A usual approach to assess the performance of a particular genetic algorithm is to monitor both the best and average fitness as the algorithm progresses. The progressive increase of the average fitness, together with the frequent improvement of the best model (indicated by the maximum fitness) is a characteristic of a genetic algorithm indicating that information obtained through previous sampling is processed efficiently to determine better data-fitting solutions. This is in contrast to random Monte Carlo, where the best-fitting model is improved on at irregular, and usually widely spaced, intervals of sampling (Sambridge and Drijkoningen, 1992; Sambridge and Gallagher, 1992). In practice, the average fitness is a more robust measure of the algorithm's performance than the best model fitness, reflecting the ability of genetic algorithms to locate rapidly regions of near-optimal solutions. As shown by Gallagher, Sambridge, and Drijkoningen, (1991), it is more meaningful to average these performance statistics for many runs before comparing one genetic algorithm implementation to another, or even to a different methodology.

Although the optimization problem considered here is trivial, we do not want to distract from the basic genetic algorithm by using a too compiex example and this serves to illustrate essential features of the implementation. However, we would reiterate that genetic algorithms are most suited to multiparameter and highly nonlinear problems, such as seismic waveform inversion. This problem has been considered with synthetic data by Stoffa and Sen (1991) and with real seismic data by Sambridge and Drijkoningen (1992). One potentially powerful use of the methodology is to generate good starting models for analytical optimization methods. For example, to be confident of converging on a reliable solution with iterative least squares, it is necessary to have a starting model which is reasonably close to the global

minimum. A genetic algorithm may be used to determine such a model rapidly.

VARIATIONS ON THE SIMPLE GENETIC ALGORITHM

As we have stated earlier, there are many variations on the basic genetic algorithm and individual problems may require different implementations to those outlined here. Variants are possible at most stages of the genetic algorithm, although the basic requirements always include the encoding, evaluation, and operation stages. Assuming that we can specify the forward calculation and the fitness function we need to consider the parameters and operators required to implement the genetic algorithm. There are no hard and fast rules regarding the selection of encoding, operators, or parameter values (e.g. number of models in population, the number of generations, or the values of the various selection probability criteria). To some extent, this initially is a drawback of the genetic algorithm approach, although experience suggests that after some experimentation for a particular class of problem, the appropriate values require little adjustment (Goldberg, 1989; Shaffer, 1989; Davis, 1989, 1991; Belew and Booker, 1991).

There are a large number of possible encodings and these can be developed simply for the most appropriate representation of the problem or in response to desirable behavior of the operators on the encoded models. For example, the mutation operator generally is desired to provide a local perturbation, yet in the simple binary encoding presented here it is equally likely that a high-order bit (i.e. a large power of 2) and a low-order bit are flipped in parity. Obviously flipping a low-order bit to the encoded model is a local perturbation to the decoded model, but the same does not apply for high-order bit. Gray coding provides one solution to this problem and in this mutation (i.e. flipping 1 bit) changes the decoded model by 1 (see Forrest, 1993). Alternatively, Sambridge and Gallagher (1993) determined that a useful variation was to change P_M depending on the bit location within a given parameter. The effect of having higher P_M for the lower power of 2 bits is to increase the ability of the algorithm to perform local searching, providing an effective method for moving off local fitness maxima, or equivalently out of local misfit minima. Davis (1990) has stated clearly that when considering an appropriate encoding (e.g. real numbers), it may be most practical and productive to incorporate the representation already in use (e.g. by another optimization method), especially when knowledge of the behavior of the problem in question can be incorporated. One simple variant that comes to mind is to develop the encoding in terms of perturbations about a given model (e.g. an a priori assumed model) rather than absolute model parameters.

The form of operators implemented in genetic algorithms differs although the basic reproduction, crossover, and mutation generally are maintained. The reproduction or selection stage discussed earlier relied on tournament selection, but there also are the roulette wheel or pie-chart methods, where the probability of a particular model being selected is determined by its relative contribution to the total or summed fitness of the population. Additional methods include selection with and without replacement of previously selected parents, elitist methods wherein the best model determined to date is replaced, and niche sharing where models which are too similar (but not equal) are penalized to increase the diversity of the parent population. These fitness-based selection methods have been used more extensively in geophysical applications than the more recently introduced rank-based tournament selection, but are prone to scaling problems. Goldberg (1989), Davis (1990, 1991), Goldberg and Deb (1991), and Forrest (1993) discuss the different selection methods in more detail.

The crossover operator can be modified to allow crossover in more than one position, to only allow crossover at the boundaries of model parameters, or to allow uniform crossover where all positions in one parent are tested for crossover and only a random selection actually are swapped with the other parent. When the ordering of the model parameters is important (not usual for earth-science problems), the crossover operator may be modified to maintain the order in the different parts of the strings. We have mentioned already a variant of the mutation operator, but many others exist. For example, when a real-number encoding is used, it can be useful to select a location in the conventional way but change the value of the digit to a randomly selected value between 0 and 9. Alternatively, the whole model may be perturbed by an amount randomly selected from within a specified range.

Clearly, there are many variants to the overall process and question arises as to what is the best form of genetic algorithm for a particular problem. There is no easy answer, although various systematic studies have been made of the sensitivity of particular problems to different parameter values (see Goldberg, 1989; Davis, 1990, 1991; Goldberg and Deb, 1991). As a general comment, however, there will be a tradeoff between the efficiency of using a small number of models in a population and a wide exploration of the model space. If the population is too small, then little sampling of the model space may occur and the algorithm may stall at a relatively poor solution (premature convergence). However, it is straightforward to examine the population for such a lack of diversity at each generation and some remedial action then may be taken. Alternatively if the population size is too large, then the time required to weed out the poorer solutions may become prohibitive. Also, if mutation is invoked too often (i.e. high mutation probability), then the algorithm will tend to be more

random and will have difficulty propagating information from one generation to the next. However, if the mutation operator and encoding are designed to provide a local search character, then a high mutation rate can be desirable. In summary, it is relatively straightforward to implement a genetic algorithm for a given problem, but it is highly recommended that some experimentation is undertaken with the parameterization to achieve the most efficient form.

CONCLUDING STATEMENTS

In this paper, we have given a brief overview of genetic algorithms and their application to optimization problems. Further details and specific discussions of refinements to the basic genetic algorithm are given in the general references we have cited, particularly Goldberg (1989), Davis (1990, 1991), Holland (1992b), and Forrest (1993), and references cited therein. In terms of optimization methodologies for nonlinear problems, genetic algorithms have many advantages over analytical techniques in that they are global, they do not require the calculation derivatives, their implementation is independent of the form of the data misfit criterion, and they provide robust sampling of the model space. Additionally, genetic algorithms differ from random Monte Carlo because they exploit sampling information through reproduction and crossover to generate better data-fitting solutions. Similar to most optimization methods, especially those applied to highly nonlinear problems, there is no guarantee that a genetic algorithm will provide an optimal solution. However, the real power of genetic algorithms lies in their ability to generate near-optimal solutions both rapidly and robustly. This characteristic makes them suitable as an optimization method in their own right, or as a method of determining good starting models for analytical methods which require initial models close to the global minimum. We believe that this powerful and rapidly evolving methodology will be applied to multiparameter, nonlinear optimization problems in many branches of the earth sciences.

REFERENCES

Belew, R. K., and Booker, L. B., 1991, eds., Proc. 4th Intern. Conf. on Genetic Algorithms and their Applications: Morgan Kaufman Associates, Los Altos, California, 576 p.

Davis, L., ed., 1987, Genetic algorithms and simulated annealing: Research notes in artificial intelligence: Pitman, London, 216 p.

Davis, L., ed., 1991, Handbook of genetic algorithms: Van Nostrand Reinhold, New York, 385 p.

Forrest, S., 1993, Genetic algorithms: principles of natural selection applied to computation: Science, v. 261, no. 5123, p. 872–878.

Gallagher, K., Sambridge, M. S., and Drijkoningen, G., 1991, Genetic algorithms: an evolution on Monte Carlo methods in strongly non-linear geophysical optimisation problems: Geophys. Res. Letts., v. 18, no. 12, p. 2177–2180.

Gallagher, K., Hawkesworth, C., and Mantovani, M., 1994a, The denudation history of the onshore continental margin of SE Brazil inferred from fission track data: Jour. Geophys. Res., in press.

Gallagher, K., Hawkesworth, C., and Mantovani, M., 1994b, Denudation, fission track analysis and the long-term evolution of passive margin topography: application to the SE Brazilian margin: Jour. South American Earth Sci., in press.

Goldberg, D. E., 1989, Genetic algorithms in search, optimisation and machine learning: Addison-Wesley, Reading, Massachusetts, 412 p.

Goldberg, D. E., and Deb, K., 1991, A comparative analysis of selection schemes used in genetic algorithms, *in* Rawlins, G. J. E., ed., Foundations of genetic algorithms: Morgan Kaufman Associates, Los Altos, California, 341 p.

Grefenstette, J. J., ed., 1985, Proc. Intern. Conf. on Genetic Algorithms and their Applications: NCARAI, Washington, D.C., and Texas Instruments, Dallas, Texas, 240 p.

Grefenstette, J. J., ed., 1987, Proc. 2nd Intern. Conf. on Genetic Algorithms and their Applications: Lawrence Erlbaum Associates, Hillsdale, New Jersey, 260 p.

Holland, J. H., 1975, Adaptation in natural and artificial systems: Univ. Michigan Press, Ann Arbor, Michigan, 183 p.

Holland, J. H., 1992a, Adaptation in natural and artificial systems: M.I.T. Press, Cambridge, Massachusetts, 221 p.

Holland, J. H., 1992b, Genetic algorithms: Scientific American, v. 267, no. 1, p. 44–50.

Jin, S., and Madriaga, R., 1993, Background velocity inversion with a genetic algorithm: Geophys. Res. Letts., v. 20, no. 2, p. 93–96.

Kennett, B. L. N., and Sambridge, M. S., 1992, Earthquake location—genetic algorithms for teleseisms: Phys. Earth Planet. Intern., v. 75, no. 1–3, p. 103–110.

Kappus, M. E., and Harding, A. J., 1993, Using genetic algorithms to produce seismic velocity models for synthetic reflection/refraction data (abst.): EOS, v. 74, no. 43, p. 393.

Nolte, B., and Frazer, L. N., 1994, VSP inversion with genetic algorithms: Geophys. Jour. Intern., v. 117, p. 162–178.

Rawlins, G. J. E., 1991, ed., Foundations of genetic algorithms: Morgan Kaufman Associates, San Mateo, California, 341 p.

Riolo, R. L., 1992, Survival of the fittest bits: Scientific American, v. 267, no. 1, p. 114.

Sambridge, M. S., and Drijkoningen, G., 1992, Genetic algorithms in seismic waveform inversion: Geophys. Jour. Intern. v. 109, no. 2, p. 323–342.

Sambridge, M. S., and Gallagher, K., 1993, Hypocentre location using genetic algorithms: Bull. Seis. Soc. America, v. 83, no. 5, p. 1467–1491.

Scales, J. A., Smith, M. L., and Fischer, T. L., 1992, Global optimization methods, for multimodal inverse problems: Jour. Comp. Phys., v. 103, no. 2, p. 258–268.

Sen, M. K., and Stoffa, P. L., 1992, Rapid sampling of model space using genetic algorithms: Geophys. Jour. Intern., v. 108, no. 1, p. 281–292.

Shaffer, J. D., ed., 1989, Proc. 3rd Intern. Conf. on Genetic Algorithms and their Applications: Morgan Kaufman Associates, Los Altos, California, 445 p.

Smith, M. L., Scales, J. A., and Fischer, T. L., 1992, Global search and genetic algorithms: Geophysics., The Leading Edge, v. 11, no. 1, p. 22–26.

Stoffa, P. L., and Sen, M. K., 1991, Nonlinear multiparameter optimization using genetic algorithms: Inversion of plane-wave seismograms: Geophysics, v. 56, no. 11, p. 1794–1810.

Whiteley, L. D., 1993, ed. Foundations of genetic algorithms 2: Morgan Kaufman Associates, Los Altos, California, 322 p.

Wilson, W. G., and Vasudevan, K., 1991, Application of genetic algorithms to residual statics estimation: Geophys. Res. Letts., v. 18, no. 12, p. 2181–2184.

Winchester, J. P., Creager, K. C., and McSweeney, T. J., 1993, Better alignment through better breeding: Phase alignment using genetic algorithms and cross-correlation techniques (abst.): EOS, v. 74, no. 43, p. 304.

Zhou, R., Tajima, F., and Stoffa, P. L., 1993, A feasibility study of genetic algorithms to constrain near-source velocity structure (abst.): EOS, v. 74, no. 43, p. 394.