Stanford Exploration Project, Report 112, November 11, 2002, pages 195–212

Can we make genetic algorithms work in high-dimensionality problems?

Gabriel Alvarez[1]

ABSTRACT

In this paper I compare the performance of a standard genetic algorithm versus a micro-genetic algorithm for matching a randomly-generated seismic trace to a reference trace with the same frequency spectrum. A micro genetic algorithm evolves a very small population that must be restarted whenever the population loses its genetic diversity. I show that the micro-genetic algorithm is more efficient in solving this problem in terms of improved rate of converge, especially in the first few generations. This characteristic may make the method useful for locating the most promising valleys in the search space which can then be searched with more traditional gradient-based methods. An additional benefit is a significant reduction in the number of evolution parameters that needs to be adjusted making the method more easy to use.

INTRODUCTION

The kind of optimization problems that we usually face in exploration geophysics are non-linear, high-dimensional, with a complex search space that may be riddled with many local minima or maxima. Usually our first line of attack is linearization of the problem around some given smooth, "easily" computed initial model. Seismic tomography is a good example of the success of this approach. There are many cases, however, where linearization is impractical or undesirable and the full non-linear problem must be solved. Broadly speaking, there are two ways to attack these kind of problems: deterministic search methods, for example non-linear conjugate gradient, quasi-Newton methods or Levenberg-Marquardt method (Gill and Murray, 1981), and stochastic search methods such as Montecarlo, simulated annealing and genetic algorithms (Davis, 1987; Goldberg, 1989a). Deterministic methods are attractive because they are natural extensions of familiar linear methods and because, in certain applications, they can be made to run extremely fast. The downside is the need to compute first and/or second order derivatives of the cost function and the dependence of the solution (and sometimes even the convergence of the method itself) on a suitable starting point. In other words, deterministic methods are extremely efficient at locating the bottom of the valley, provided they start the search somewhere inside the valley. This is a serious shortcoming since in many problems

[1]**email:** gabriel@sep.stanford.edu

locating the valley that contains the minimum may be a problem as difficult as locating the minimum itself. We could say that deterministic methods are poor at "exploration" (locating the best valleys) but are very good at "exploitation" (given the valley, locating its floor).

Stochastic methods, on the other hand, perform a much more exhaustive search of the model space but are not as good at exploiting the early results of the search. It appears that a hybrid method combining the strengths of both techniques would be the best choice. The situation is not so clear cut, however, because it may be difficult to identify the most promising valleys and it may also be difficult to compute the derivatives of the cost function. In geophysics such a hybrid approach between a genetic algorithm and conjugate gradient was used to solve the problem of estimating velocities from refraction seismic data, although the results were not conclusively better than those obtained by the genetic algorithm alone (Boschetti, 1995).

An interesting alternative is the use of the so-called micro-genetic algorithms (Krishnakumar, 1989) which aim at improving the relatively poor exploitation characteristic of the genetic algorithms without affecting their strong exploration capabilities. In this paper I compare the results of applying both a standard and a micro-genetic algorithm to the problem of matching a seismic trace. This is part of the more interesting problem of inverting a zero-offset trace for interval velocities addressed in a companion paper in this report (**?**). Figure 1 shows the input sub-sampled sonic log and the corresponding reference seismic trace obtained by a simple computation of the normal incidence reflection coefficients assuming no multiples, no absorption and no noise.

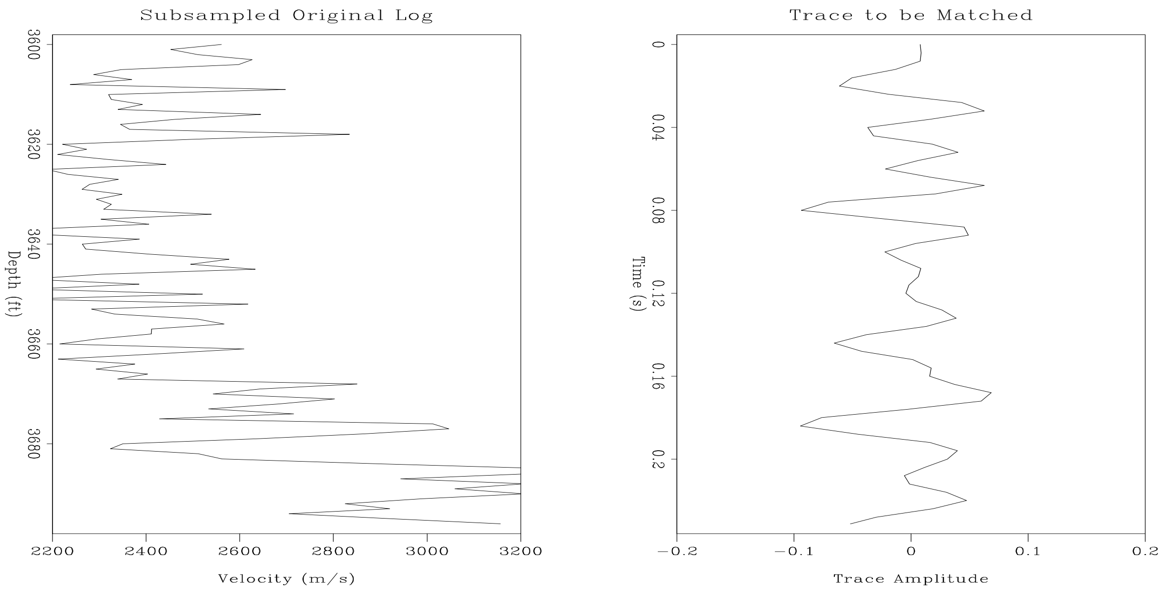

Figure 1: Left, sub-sampled sonic log used to generate the synthetic seismic trace on the right gabriel1-input [ER]

STANDARD GENETIC ALGORITHM

A genetic algorithm is an optimization method inspired by evolution and survival of the fittest. A trial solution to the problem is constructed in the form of a suitably encoded string of model parameters, called an *individual*. A collection of individuals is in turn called a *population*. There are several considerations and choices to be made in order to implement a suitable solution to an optimization problem using genetic algorithms. A full description of all the practical details is outside the scope of this paper, and some of them are a matter of active research (Gen and Cheng, 2000; Haupt and Haupt, 1998; Falkenauer, 1998; Beasley et al., 1993). In Appendix A I give a brief description of the most relevant issues of genetic algorithm optimization as used in this study. In particular, I describe model-parameter encoding as well as standard and non-standard operators (selection, jump and creep mutation, crossover, elitism and niching), fitness function and convergence criteria.

Parameter Selection

The first step, of course, is to select the *model* parameters related to the "physics" of the problem, their ranges (maximum and minimum values) and their required resolution. Then it is time to choose the *evolution* parameters actually related to the genetic algorithm itself. The performance of a genetic algorithm to solve a particular optimization problem depends critically on the choice of its evolution parameters that must be fine-tuned to that problem as much as possible. In general it is difficult to give hard and fast rules that may work with a wide range of applications, although some guidelines exist (Goldberg and Richardson, 1987; Goldberg, 1989b; Goldberg and Deb, 1991). For this problem, I choose the evolution parameters one by one starting with the most critical and working my way down to the least critical. Once a particular parameter is selected it is kept constant in the tests to select the remaining parameters. I do so because it would be nearly impossible to test all possible combinations. The reader may find useful to refer to Appendix A for a description of the evolution parameters themselves.

Model Parameter Encoding

For the present application I use binary encoding of the 99 *model* parameters (not to be confused with the *evolution* parameters that control the inner workings of the genetic algorithm) representing potential solutions to the problem (**?**). I represent a model parameter with 10 bits so that there are $2^{10} = 1024$ possible values for each model parameter and a total of 1024^{99} possibilities for the entire search space. A completely exhaustive search of the model space would therefore be very difficult if not impossible.

Population Size

Ideally, the population size should be large enough to guarantee adequate genetic diversity yet small enough for efficient processing. In particular, the number of cost-function evaluations is proportional to the population size. Equation 1 corresponds to Goldberg's criterion (Goldberg, 1989b) of increasing the population size exponentially with the increase in the number of model parameters (assuming binary encoding).

$$npopsize = order[(l/k)(2**k)] \tag{1}$$

In this equation, l is the number of bits in the chromosome and k the order of the schemata of interest (schemata is plural for schema [2]). This criterion, however, may result in populations too large when the number of model parameters and so the length of the chromosome is large. In this case, for example, with 99 model parameters, each encoded with 10 bits, even for a relatively low-order schema of 5 bits the population size would have to be larger than 3000 individuals. Such large populations may require many cost-function evaluations and a lot of memory. Most applications reported in the literature use population sizes between 50 and 200 individuals and I am unaware of any reported use of a population larger than 1000 individuals. For the purpose of the present application I decided to try a relatively wide number of population sizes using for the other *evolution* parameters "reasonably" standard choices. For example, I used uniform crossover with a probability of 0.6, jump mutation probability equal to $1/npopsize$ (large populations have larger genetic diversity and so less need for jump mutation) and creep mutation probability equal to the number of bits per model parameter times the jump mutation rate. Also, for this first set of tests I allowed both niching (sharing) and elitism of the best individual.

The top panels of Figure 2, from left to right, show a comparison of convergence rate as a function of number of generations for population sizes of 50, 100, 200 individuals whereas the bottom panels show similar curves for population sizes of 250, 500 and 1000 individuals. As expected, the larger populations produce better convergence after a fixed number of generations. This is not a fair test, however, since the number of cost-function evaluations is proportional to the population size. Figure 3 shows the same curves for a fixed number of function evaluations. Clearly, there is a practical range of optimum population sizes about 200 or 250 individuals. For all the following tests I used a population of 200 individuals. Figure 4 shows a comparison of the reference trace (solid line) with the inverted traces (dotted line) obtained with each population in the same order as that in Figure 3. The difference in the match of the traces is not so impressive because in all cases a good solution is eventually found for all populations. The smallest population, however, does show a poorer match than the others.

Selection Mechanism

I used a tournament selection in which each parent is the best fit of two individuals picked at random from the population. This technique has the advantage of applying significant selection pressure while avoiding the pitfalls of fitness ordering or ranking (Falkenauer, 1998).

[2] A schema is a similarity template describing a subset of strings with similarities at certain string positions. For example, the schema 0*1 matches the two strings 001 and 011.

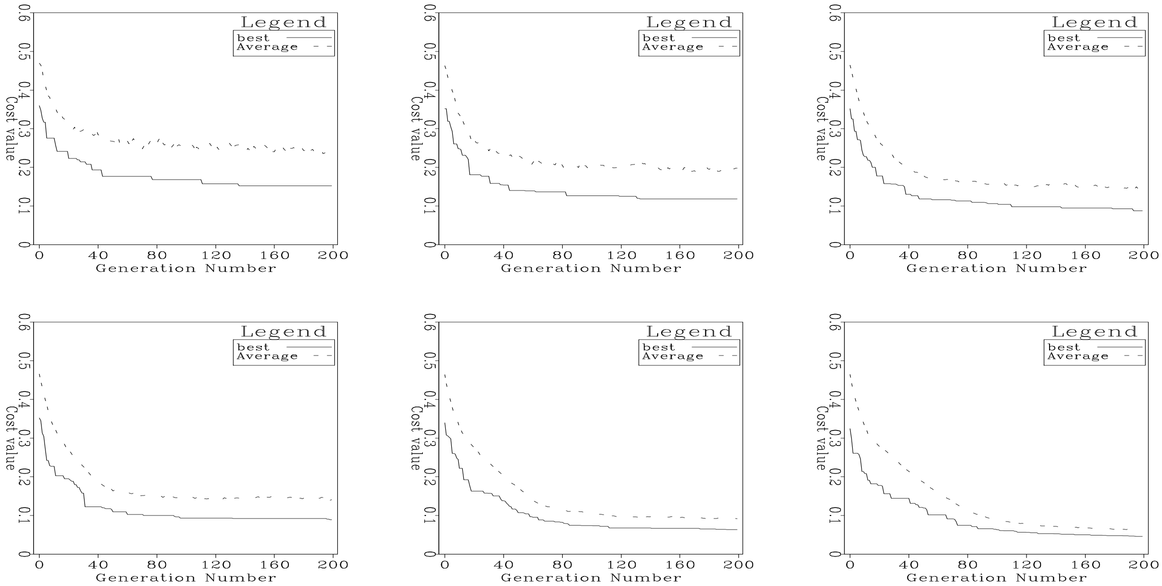

Figure 2: Comparison of convergence rates for different population sizes keeping the number of generations constant. Top row populations 50, 100 and 200 individuals. Bottom row, populations of 250, 500 and 1000 individuals. gabriel1-SG_compare_pop_sizes1 [ER]

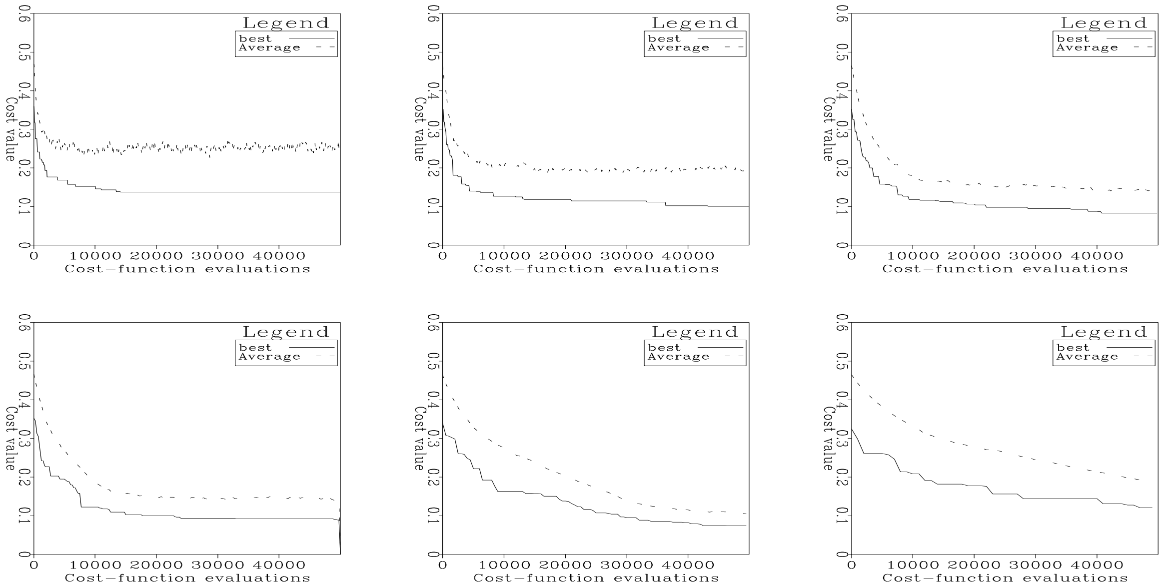

Figure 3: Same as in Figure 2 but in terms of number of function evaluations rather than number of generations. gabriel1-SG_compare_pop_sizes2 [ER]

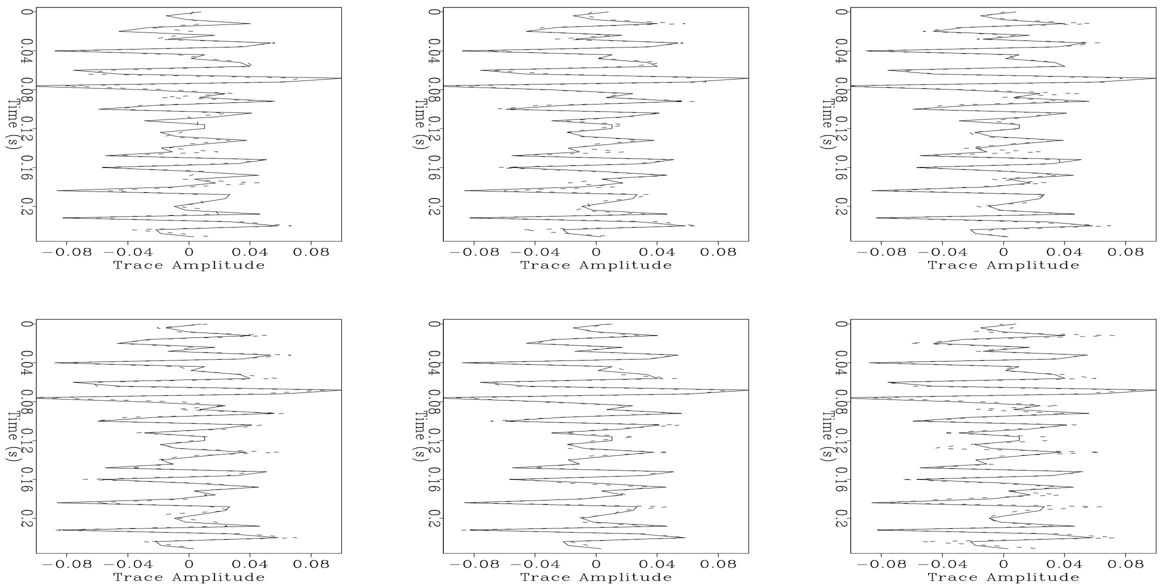

Figure 4: Comparison of trace match for different population sizes keeping the number of function evaluations constant. Continuous line is the reference trace and dotted line the inverted trace. Top row, populations of 50, 100 and 200. Bottom row, populations of 250, 500 and 1000. gabriel1-SG_compare_tr_pop_sizes2 [ER]

Crossover Rate

Crossover is by far the most important evolution operation. I tested single-point and uniform crossover with a crossover probability of 0.6. The population size was chosen to be 200 and the algorithm was run for 250 generations. The jump mutation was set at 0.005 and the creep mutation at 0.05. Elitism of the best individual as well as niching was allowed. The top panels in Figure 5 show the convergence rate of the two cases, whereas the bottom panels show the corresponding traces. In this case uniform crossover (right panel) performs a little better since it reaches a lower cost-value after the allowed number of cost-function evaluations. Using uniform crossover, I tried six different values of crossover probability: 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. I tried this large range of values because crossover rate is a particularly important evolution parameter. The top panels of Figure 6 show the comparison of the convergence rates for crossover rates of 0.5, 0.6 and 0.7 whereas the bottom panels show the same curves for crossover rates of 0.8, 0.9 and 1.0. The results are surprisingly similar, although it appears that the smaller crossover rates produce faster initial convergence, and so I chose a crossover rate of 0.6 for the remaining tests.

Mutation

As mentioned in Appendix A, there are two types of mutation operators: the standard jump mutation that acts on the chromosome (binary representation of the individual, sometimes called *genotype*) and creep mutations that act on the decoded individual, sometimes called

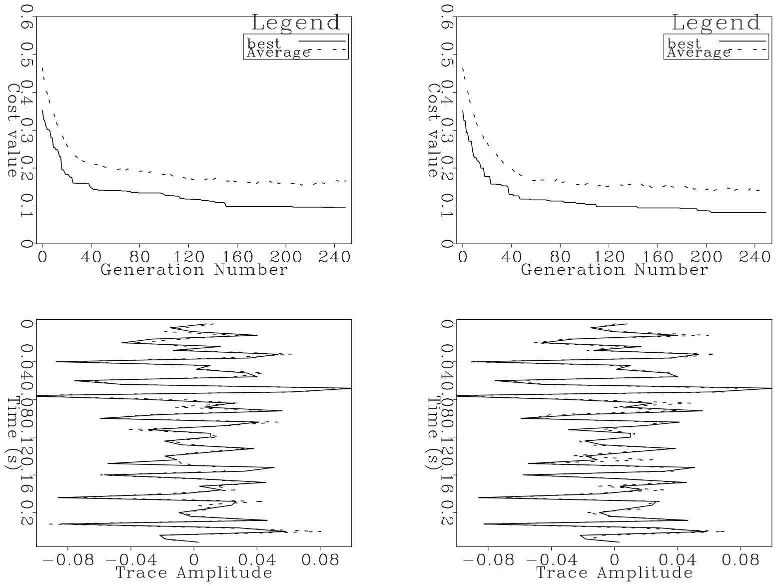

Figure 5: Comparison of convergence rates for two types of crossover: single-point (left) and uniform (right). Top panels are convergence rates whereas bottom panels are trace match with continuous line representing the reference trace and dotted line the inverted trace. gabriel1-SG_compare_crossover1 [ER]

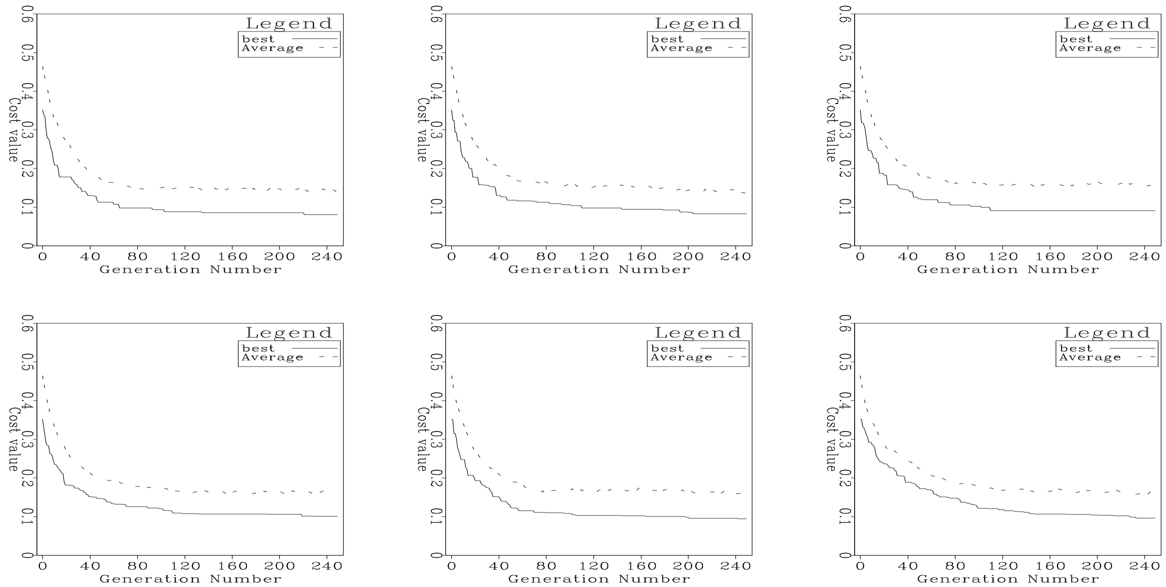

Figure 6: Comparison of convergence rates for different crossover rates. Top panels correspond to crossover rates of 0.5, 0.6 and 0.7 whereas bottom panels correspond to crossover rates of 0.8, 0.9 and 1.0. gabriel1-SG_compare_crossover2 [ER]

phenotype. In any case, the mutation probabilities are expected to be low, since high values may cause strong disruption of promising schemata and therefore steer the algorithm away from the most promising regions of the search landscape. The top panels of Figure 7 show a comparison of convergence rates for three values of jump mutation probability 0.002, 0.004 and 0.008, without creep mutations. In this case a jump mutation of 0.002 is the best although a value of 0.005 would have been expected from the rule of thumb of the inverse of the population size. The bottom panels of Figure 7 show the effect of creep mutations with probabilities of 0.02, 0.04 and 0.08. Again, it seems that a small mutation is actually best. For the remaining I used creep mutation with probability 0.02.

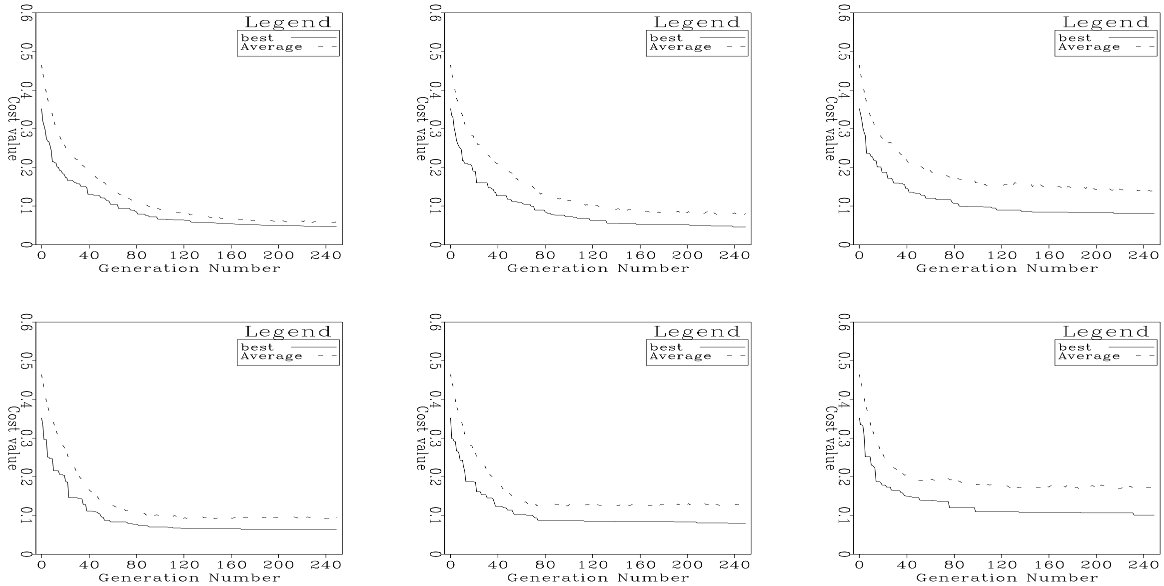

Figure 7: Comparison of convergence rates for different jump and creep mutation rates. Top panels for jump mutation rates of 0.002, 0.004 and 0.008. Bottom panels for creep mutation rates of 0.02, 0.04 and 0.08. gabriel1-SG_compare_mutation1 [ER]

Other options

Finally, I tested the inclusion of niching (Goldberg and Richardson, 1987) and elitism. The top panels of Figure 8 compare the convergence rates with niching and elitism (left)i, only elitism (middle), and only niching (right). The bottom panels show a similar comparison for the trace match. All the other evolution parameters were chosen according to the previous analysis. It seems that the lack of niching makes the convergence a little slower in the first iterations (compare the left and middle panels) but makes it faster after about the 40th generation. The lack of elitism, on the other hand, makes convergence a little erratic (compare left and right panels) since we are not guaranteed to go to a better-fit best individual in any generation compared with the previous one. Elitism was thus included and niching excluded in the evolution parameters of the final optimum standard genetic algorithm.

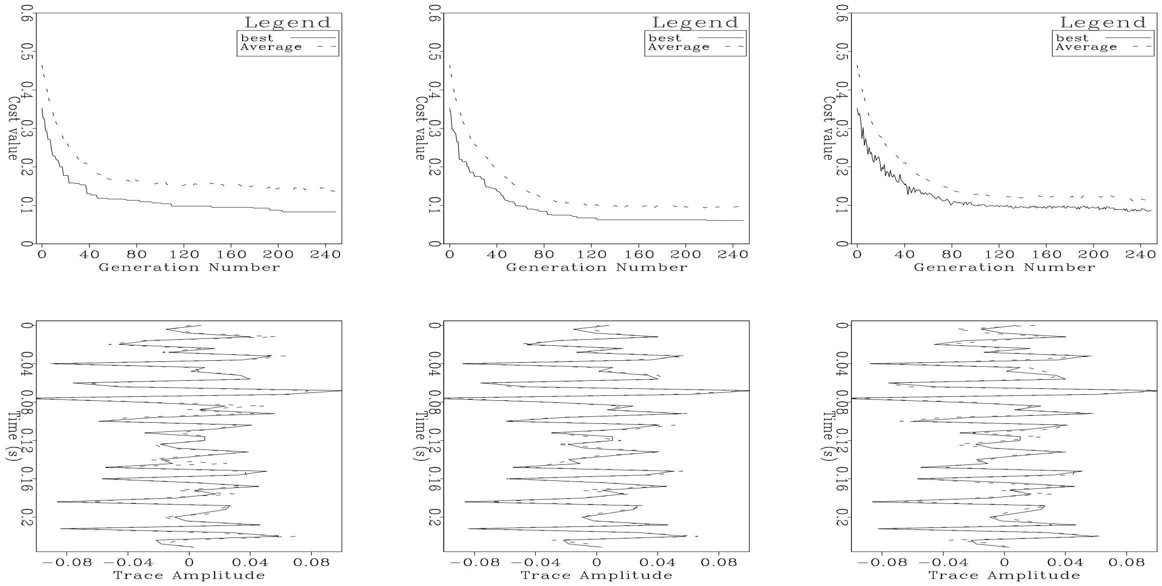

Figure 8: Comparison of results for the inclusion of niching and elitism. Top panel convergence rates: left with niching and elitism, middle with elitism only and right with niching only. Bottom panels show similar comparison for the trace match with continuous line representing the reference trace and dotted line the inverted trace. gabriel1-SG_compare_other1 [ER]

Parameter Summary

For easy reference, Table 1 shows the evolution parameters selected as optimum from the previous tests. The resulting genetic algorithm will be compared with a micro-genetic algorithm described in the next section.

Table 1: Summary of the optimum evolution parameters for the standard genetic algorithm

Population size	200	Crossover rate	0.6
Mutation rate	0.002	Creep mutation rate	0.02
Elitism	Yes	Niching	No
Selection strategy	Tournament	Number of children	1

MICRO-GENETIC ALGORITHM

When dealing with high dimensionality problems, it may be difficult or too time consuming for all the model parameters to converge within a given margin of error. In particular, as the number of model parameters increases, so does the required population size. Recall that large population sizes imply large numbers of cost-function evaluations. An alternative is the use of micro-genetic algorithms (Krishnakumar, 1989), which evolve very small populations that are very efficient in locating promising areas of the search space. Obviously, the small populations are unable to maintain diversity for many generations, but the population can be restarted whenever diversity is lost, keeping only the very best fit individuals (usually we keep just the best one, that is, elitism of one individual). Restarting the population several times during the run of the genetic algorithm has the added benefit of preventing premature convergence due to the presence of a particularly fit individual, which poses the risk of preventing further exploration of the search space and so may make the program converge to a local minimum. Also, since we are not evolving large populations, convergence can be achieved more quickly and less memory is required to store the population.

Selection of Evolution Parameters for Micro-GA

In principle, micro-genetic algorithms are similar to the standard genetic algorithm described in the previous section, in the sense of sharing the same evolution parameters and similar considerations. There is, however, an important distinction: since new genetic material is introduced into the population every time the algorithm is restarted, there is really no need for either jump or creep mutation. Also, elitism is required, at least every time the population is restarted, otherwise the algorithm would lose its exploitation capability. I have also found that the algorithm is much less sensitive to the choice of evolution parameters compared with the standard genetic algorithm. In particular, population sizes of 5 to 7 with crossover rates of 0.8 to 0.95 give very good results. The top panels of Figure 9 shows a comparison of convergence rates for populations of 3, 5 and 7 individuals. It seems clear that 5 individuals is the best. This result agrees with Carroll's who employed micro-GAs to optimize an engineering problem (Carroll, 1996). The bottom panels show a comparison of convergence rates for populations of 5 individuals and crossover rates of 0.7, 0.9 and 1.0. It seems that 0.9 is the best crossover rate, although further tests showed that 0.95 gave even better results and therefore that value was chosen for the remaining tests. Figure 10 shows the results in terms of trace match. Again, it is apparent that a population of 5 and a crossover rate of 0.9 are optimum. In particular, note how a uniform crossover of 1.0 (bottom right panel) is far too disruptive.

Summary of Evolution Parameters for Micro-GA

Table 2 shows a summary of the evolution parameters selected for the micro-GA for the current application.

COMPARISON OF STANDARD AND MICRO-GA

Having chosen the evolution parameters that provided the best results for both the standard and the micro-GA (Tables 1 and 2), I will now compare the performance of the two in solving the current problem.

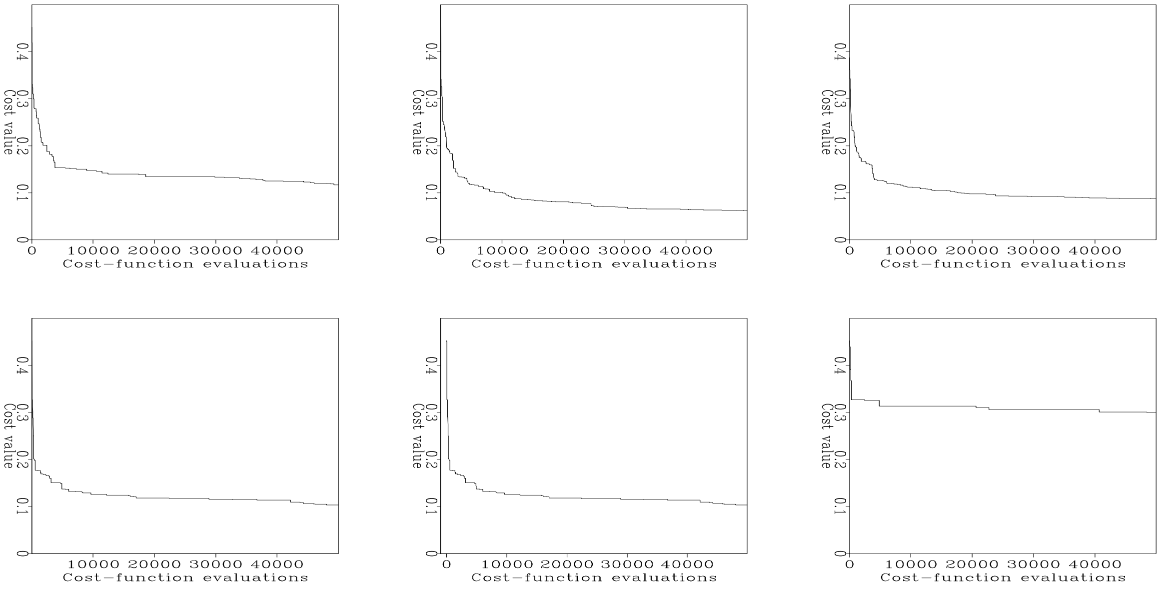

Figure 9: Comparison of convergence rates for different options of the micro genetic algorithm. Top panel population rates of 3, 5 and 7 (from left to right). Bottom panels, with population size of 5 and crossover rate of 0.7, 0.9 and 1.0. gabriel1-MG_compare1 [ER]

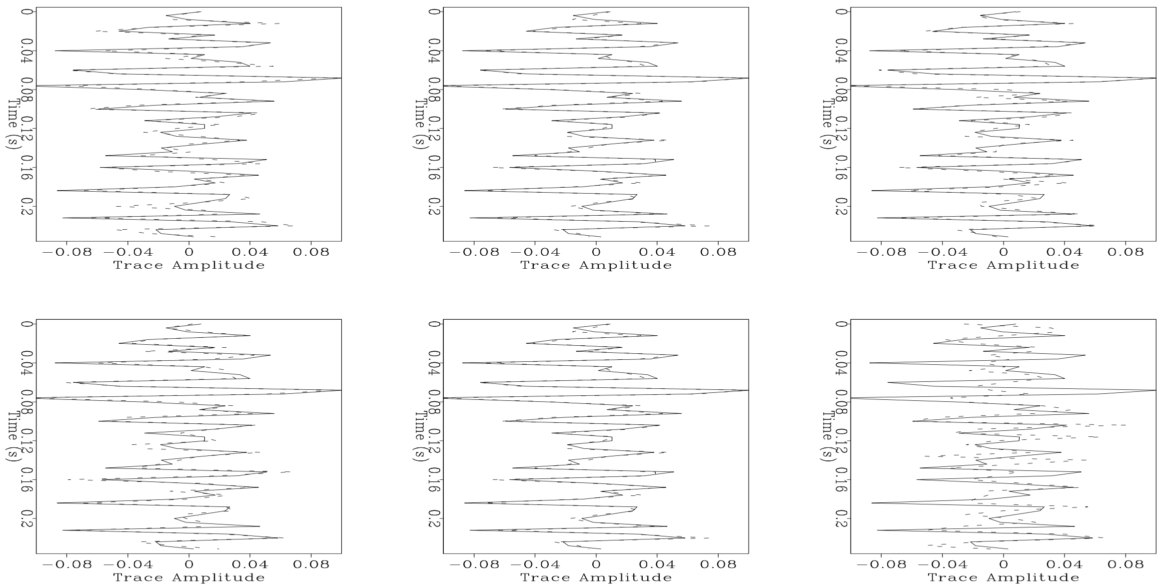

Figure 10: Comparison of trace match for different options of the micro genetic algorithm. Continuous line is the reference trace and dotted line is the inverted trace. Top panel population rates of 3, 5 and 7 (from left to right). Bottom panels, with population size of 5 and crossover rate of 0.7, 0.9 and 1.0. gabriel1-MG_compare2 [ER]

Table 2: Summary of the optimum evolution parameters for the micro-genetic algorithm

Population size	5	Crossover rate	0.95
Jump mutation rate	0.0	Creep mutation rate	0.0
Elitism	Yes (best individual only)	Niching	No
Selection strategy	Tournament	Number of children	1

The top panels of Figure 11 show a comparison of convergence rates between the standard genetic algorithm (left) and the micro-genetic algorithm (right). The difference in convergence rate is in the first generations is impressive. For example, the micro-genetic algorithm would have essentially converged after 2000 cost-function evaluations, whereas the standard genetic algorithm would take almost 10000 cost-function evaluations to reach the same convergence level. If enough iterations (generations) are allowed both algorithms will converge to essentially the same result. The bottom panels in Figure 11 show the corresponding trace match. The differences are not too great because both algorithms were essentially run to convergence.

CONCLUSIONS AND FUTURE WORK

The results of this test are encouraging because micro genetic algorithms show a much faster rate of convergence than standard genetic algorithms in the solution of this simple, relatively high-dimensional problem. At the very least micro genetic algorithms could be run for a few generations and use the results as starting points for gradient-based methods.

From the point of view of the genetic algorithm inversion, some lessons have been learned after extensive testing of the evolution parameters. Firstly, using a micro-genetic algorithm with uniform cross-over without mutation emerges as the best option for this problem (as opposed to a standard genetic algorithm with single-point cross-over and jump and creep mutation). Secondly, a micro-genetic-algorithm population of 5 individuals with a cross-over probability of 0.95 seems to be optimum for this problem.

An important issue to be further analyzed is that of the multi-modality of the search space. In this case it is clear that there is a single global minimum, namely recovering the original trace sample-by-sample. However, I have found that once I get close enough to this global minimum it takes a large number of iterations to escape local minima (many traces "almost fit" exactly the original). My present convergence criteria do not allow for checking of convergence of individual model parameters so I have to investigate alternative options.

Another important issue has to do with the convenience of working with the model parameters directly in their floating-point representation rather than the standard binary encoding used here. This approach has the advantage of not requiring a resolution limit on the model parameters.

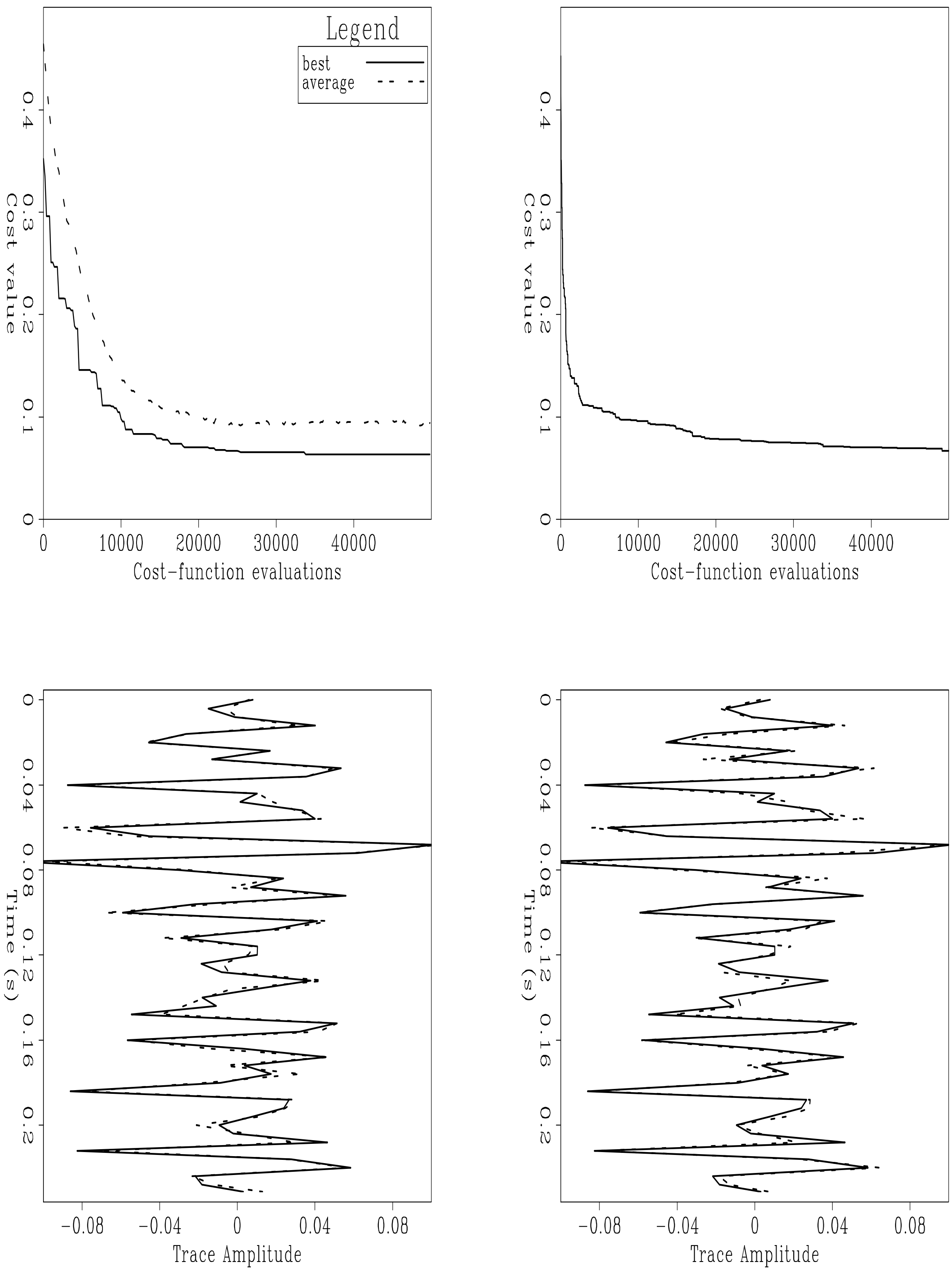

Figure 11: Comparison between the standard and the micro genetic algorithm. Top panels convergence rates for standard genetic algorithm (left) and micro genetic algorithm (right). Bottom panels show the corresponding trace match with continuous line representing the original trace and the dotted line the inverted ones. gabriel1-final_comparison [ER]

APPENDIX A: REVIEW OF GENETIC ALGORITHMS

In this appendix I briefly review some of the terms and issues related to genetic algorithms in optimization. More detailed accounts can be found in (Goldberg, 1989a; Haupt and Haupt, 1998; Falkenauer, 1998; Gen and Cheng, 2000).

Model Parameter Encoding

The choice of representation of the problem in terms of how many model parameters, their encoding, range of values and required resolution is perhaps the most important decision we face when using genetic algorithms. In particular, we must decide whether to use "direct" representation of the problem (in terms of floating point numbers,for example) or to "encode" the solution in terms of a suitable "alphabet", usually binary. In his pioneering work in genetic algorithms Holland employed the binary representation to prove his schemata theorem which provides the theoretical foundation for the workings of the genetic algorithm (Holland, 1975; Goldberg, 1989a). This theorem proves that short, low order schemata are more likely to be preserved by the evolution process in contrast to long high order schemata which are more likely to be disrupted by crossover and mutation [3]. Short, low order schemata, therefore, are likely to end up associated with highly fit individuals, that is, with the best solutions to the problem. Therefore, if we can encode our model parameters in such a way that the promising short low-order schemata are produced, we may achieve a faster convergence and obtain a better solution. If we use binary encoding, in order to have short, low-order schemata, the model parameters must be suitably encoded with related model parameters being put close together in the binary string representing an individual (Goldberg, 1989a). The problem is that in general we may not know before hand which parameters are related to others or to what extent they are related. Therefore, in general it is difficult to establish the order of the predominant schemata in a given encoding of the model parameters.

Basic Operators

A genetic algorithm optimization begins by generating a population of randomly computed individuals within the constraints imposed on the model parameters. Three basic operators: reproduction (selection), crossover and mutation are used to "evolve" the solution from one generation (iteration) to the next.

Selection

Each trial solution (individual) is assigned a figure of merit that represents how good a solution it is according to the fitness function (cost function or objective function) of the problem. The

[3]The order of a schema is the number of fixed positions. For example, the order of 011*1** is 4. The length of a schema is the distance between the first and the last specific string positions. For example, the schema 011**1* has a length of 6-1=5

most fit individuals (lowest cost-values for minimization problems) are given a higher probability of mating in order to produce the next generation. There are several ways to select the "parents" for mating, such as random pairing, roulette wheel, rank weighting and tournament selection (Haupt and Haupt, 1998). Whatever the selection method, the net effect is to skew the next generation towards the most fit individuals, that is, towards the most promising regions of the search space. It is still possible and desirable to allow less fit individuals to mate, albeit with a lower probability. This increases the exploration of the search space and helps prevent premature convergence, i.e. convergence to a local minimum.

Crossover

Crossover is the operator actually responsible for the exchange of genetic material between the parents in order to produce their offspring. In the usual case of binary encoding, the simplest crossover operator randomly selects a bit position in the binary string representing an individual (a chromosome), and then two children are produced by taking the bits to the left of the crossover point in parent one and those to the right in parent two and vice-versa. This is called single-point crossover. More than one point may be selected, two, for example, such that one child consists of the bits from the two points to the ends in one parent and those in between the two points from the other parent. The other child will be similarly produced by exchanging the role of the two parents. In the limit a child may be produced by randomly selecting, for each bit, the value from one or the other parent. This is called uniform crossover. It is important to notice that crossover is not necessarily applied to all couples, but only according to a given probability, usually between 0.5 and 0.9. Also, when using other encodings, for example floating-point numbers, the operator must be adjusted (Haupt and Haupt, 1998).

Mutation

The operator responsible for introducing new genetic material into the population is called mutation (or more precisely jump mutation). With binary encoding, this operator works by randomly selecting a bit and then flipping it. In general, mutation is applied with a relatively low probability, usually less than 0.1. The idea is that although mutations are critical to prevent the population from loosing their genetic diversity (that is, mutations force the algorithm to look into other areas of the search space) they are potentially disruptive causing the algorithm to lose information from a promising search area.

Other operators

There are many other operators that may be used to improve the chances of a fast and accurate convergence of genetic algorithms. The ones used in this study are

- Elitism: This is the operator that promotes, without change, the best individual or individuals of the population to the next generation. These individuals are still eligible for recombination with others to produce the offspring.

- Niching: Also called sharing is the process of sharing genetic material between closely related individuals (Goldberg and Richardson, 1987)

- Creep mutation: Similar to standard mutation but only changes that result in small perturbations of the decoded model parameters (as opposed to their binary representation) are allowed. This can be considered a fine-tuning operator.

Fitness function

The fitness function is the equivalent to the cost function in standard optimization theory. In a minimization problem the fitness of each individual is evaluated and a figure of merit assigned such that the individuals with the smallest cost-function values are considered the most fit.

Convergence

In general, establishing the convergence of a genetic-algorithm optimization may not be an easy matter because we do not know for sure if the algorithm has converged to a local or global minimum. In the first case we would like the algorithm to continue exploring the search space and in the second case we would like the algorithm to stop. There are several different ways in which we can proceed, for example:

- Set a threshold for the minimum cost function that constitutes an acceptable solution and stop the algorithm when this value is reached. The problem with this approach is that in some cases it may be too difficult to establish a priori what a good threshold should be.

- Set a threshold for the *difference* between the best and the average fit individual in the population. This has the advantage of not requiring a priori knowledge of the actual cost function values.

- Set a threshold for the difference between the best individuals of the present and the previous generation or generations. This has the advantage of preventing the algorithm from attempting to refine the search too much by slowly crawling to the bottom of the valley.

- Set a maximum number of generations to be evolved. This is a fail safe criterion which can be useful when comparing the performance of genetic algorithms with different combination of evolution parameters.

In most cases a combination of two or more of these and similar criteria are employed.

REFERENCES

Beasley, D., Bull, D., and Martin, R., 1993, An overview of genetic algorithms: Part 2, research topics: University Computing, **15**, no. 4, 170–181.

Boschetti, F., 1995, Application of genetic algorithms to the inversion of geophysical data: Ph.D. thesis, The University of Western Australia.

Carroll, D., 1996, Genetic algorithms and optimizing chemical oxygen-iodine lasers: Developments in theoretical and applied mechanics, **18**, 411–424.

Davis, L., 1987, Genetic algorithms and simulated annealing.: Pitman.

Falkenauer, E., 1998, Genetic algorithms and grouping problems: John Wyley and sons.

Gen, M., and Cheng, R., 2000, Genetic algorithms and engineering optimization: John Wyley and sons.

Gill, P., and Murray, W., 1981, Practical optimization: Academic Press.

Goldberg, D., and Deb, K., 1991, A comparative analysis of selection schemes used in genetic algorithms: A comparative analysis of selection schemes used in genetic algorithms:, Morgan Kaufmann, Foundations of Genetic algorithms.

Goldberg, D., and Richardson, J., 1987, Genetic algorithms with sharing for multimodal function optimization: Genetic Algorithms and their application. Proceedings of the Second International Conference on Genetic Algorithms, Morgan Kaufmann Publishers.

Goldberg, D., 1989a, Genetic algorithms in search for optimization and machine learning: Addison-Wesley Pub. Co.

Goldberg, D., 1989b, Sizing populations for serial and parallel genetic algorithms: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers.

Haupt, R., and Haupt, S., 1998, Practical genetic algorithms: John Wyley and sons.

Holland, J., 1975, Adaptation in natural and artificial systems: University of Michigan Press.

Krishnakumar, K., 1989, Micro-genetic algorithms for stationary and non-stationary function optimization.: SPIE: Intelligent control and adaptive systems, **1196**, 289–296.