Employing Gradient Information in a Genetic Algorithm

Antti Autere
Helsinki University of Technology, Department of Computer Science
Otakaari 1 A, SF-02150 Espoo, Finland.
email: aau@cs.hut.fi

Abstract

This paper presents an algorithm for unconstrained optimization of differentiable functions with many local optima. It combines genetic algorithm (GA) and gradient search techniques in a novel way. Instead of one population of solution candidates it has two. One is sorted by the criteria function values and the other by the lengths of its gradients.

The proposed algorithm has been compared to a one-population genetic algorithm with and without a gradient search. The test functions are taken from a family of modified Shekel's functions. One thousand 10 dimensional test functions with 2-20 randomly located minima have been minimized by the tested algorithms. The results have been collected into histograms where one may visually observe the effects of included gradient information.

The reference [AAK94] presents the distributed version of the algorithm.

1 Introduction

This paper extends the work we have done in combining the genetic algorithm and gradient search [Aut94], [AAK94]. The earlier simulation results from [Aut94] encourages to make use of gradient information whenever it is possible to get.

Calculating one gradient by finite differences costs n criteria function evaluations, where n is the dimension of the function. We think that once the gradient has been obtained for the purpose the line search routine it may be of advantage to store it for possible later use. How could one include this information into the structure of the GA ?

We use the rank-based selection of parent candidates from the population: the population is sorted by the fitness values and the best ones are selected for further use.

In addition to this we include the gradients and the lengths of the gradients into the population. So the individuals of the population would be quadruples:

$$(\mathbf{x}, f(\mathbf{x}), \nabla f(\mathbf{x}), - \| \nabla f(\mathbf{x}) \|), \tag{1}$$

where $\mathbf{x}$ is a solution candidate, $f(\mathbf{x})$ its function value and $- \| \nabla f(\mathbf{x}) \|$ the opposite of the length of its gradient.

Actually, we have a population with two objectives: the function value and the length of the gradient. The population can only be ordered *partially* by the two criteria. The solution $\mathbf{x}$ is better than $\mathbf{y}$ (or *dominates* $\mathbf{y}$) if and only if its *both* objective values are better than $\mathbf{y}$'s [Coh78] (p.77) i.e.

$$(f(\mathbf{x}) \geq f(\mathbf{y})) \wedge (g(\mathbf{x}) \geq g(\mathbf{y})) \tag{2}$$

and at least the one $\geq$ is $>$. If this is not the case no unique ordering between the individuals exists. How would one select the best candidates ?

2 Multiobjective Populations

In the GA literature several ways for sorting the multiobjective population have been presented. Let us mention a few here. Schaffer forms sub-populations generated by performing proportional selection from the current population according to each objective in turn. These would then be shuffled together to obtain a new population [Sca85], [SG85].

Fonseca and Fleming use dominance to sort the population [1] [FF93]. The individual y has a rank number: 1 + the number of the individuals that dominate it in a sense of the equation (2). Goldberg presents a somewhat different sorting mechanism also based on dominance [Gol89] (p. 201).

Louis and Rawlins utilize dominance at the mating -level. They use a variant of binary tournament selection [LR93]. Their algorithm selects two individuals at random from the current population, mates them, and produces the set from the parents and offspring where no one dominates another (the *pareto optimal* set). Two random individuals from this set form part of the next population. The procedure repeats until the new population fills up.

The problem on our case is that the gradients are much more expensive to get than the single criteria function values. So, the gradients are not calculated every time a new solution (an individual) is produced. This means that most of the quadruples of the equation (1) have nothing in their gradient slots. We think that trying to sort this population in some way would not give enough weight to the gradients and this weight would not be in control.

Instead, we have *two separate populations* sorted by their own objective: the function value and the length of the gradient. Before selecting the parents the algorithm chooses the population from which they will be picked. After that the rank-based selection is used to find the parents. Note that the intersection of the two populations may not be empty.

Finally, we are minimizing a single function and use multiobjective population as a tools for it. We believe that this is a new idea in the context of using gradient information. Louis and Rawlins have a similar idea for bit-string function optimization [LR93].

3 The Algorithms Tested

The following pseudo code illustrates the algorithm. Let `P` and `DP` stand for the populations sorted by the function values and the lengths of the gradients, respectively. Further, let `SP` be the selected population and `p1`, `p2` the parents in `SP`. `c` is the child. `Rnd` is uniformly distributed (0,1).

```
initialize_populations(P,DP);
iter := 1;
WHILE (iter < max_iter)
      rnd = uniform(0,1);
      IF (0 <= rnd < 0.05) THEN mutate(c);
      ELSE
           select_population(P,DP,SP);
           select_parent(SP,p1);
           select_population(P,DP,SP);
           select_parent(SP,p2);
           IF (0.05 <= rnd < 0.5) THEN crossover(p1,p2,c);
           IF (0.5 <= rnd <= 1.0) THEN line_search(p1,p2,c);
      END
      update_populations(c,P,DP);
      iter := iter + 1;
END
```

Initialize-populations generates a random starting population. The gradient population will be empty.

[1] The population now has n objectives

Select-population selects either one of the populations `P` or `DP` at random. The gradient population `DP` is selected only if it contains enough individuals (5). *Select-parent* selects one parent from the population `SP` at random.

The algorithm uses the ranking selection scheme (*update-populations*). A new member is accepted only if its fitness is less than the fitness of the last individual in the list.

Crossover produces a child $\mathbf{c} \in R^n$ from two parents $\mathbf{p}_1, \mathbf{p}_2 \in R^n$ according to a formula:

$$\mathbf{c} = \mathbf{R} * \mathbf{p}_1 + (\mathbf{I} - \mathbf{R}) * \mathbf{p}_2, \tag{3}$$

where $\mathbf{R}$ is a diagonal matrix, $\mathbf{R} = diag(r_1, ..., r_n)$, whose elements are uniformly distributed (0,1). $\mathbf{I}$ is the identity matrix. We may think that the child is selected from the hypercube whose opposite corners the parents are. Let us call the crossover *box*. [2]

Mutate generates an individual $\mathbf{c} \in R^n$ at random.

3.1 The Algorithm without Gradients: GA

GA has only one population. In addition to mutation and crossover GA has a line search routine. *Line-search* produces a child $\mathbf{c} \in R^n$ from the parents $\mathbf{p}_1, \mathbf{p}_2 \in R^n$ according to a formula:

$$\mathbf{c} = r * \mathbf{p}_1 + (1 - r) * \mathbf{p}_2, \tag{4}$$

where r is a real number from the uniform distribution (0,1). So, the child is selected from the line joining the parent vectors.

The simulation results of this algorithm are from [Aut94]

3.2 GA with Golden Section Gradient Search: GAG

The algorithm is the same as GA but the *line-search* is replaced by a *golden section line search* along the direction of the negative gradient, see e.g. [BS79] p. 259.

$$\mathbf{c} = \mathbf{p1} - t(\| \mathbf{p1} - \mathbf{p2} \|) * \frac{\nabla f(\mathbf{p1})}{\| \nabla f(\mathbf{p1}) \|}, \tag{5}$$

where $\mathbf{p1}$ and $\mathbf{p2}$ are parents, t is a step size and f is the function to be minimized. It is also checked that both the parents $\mathbf{p1}$ and $\mathbf{p2}$ are not the same. $t = uniform(min, \| \mathbf{p1} - \mathbf{p2} \|)$ or $t = uniform(min, max)$, where $min = max/200$.

The golden section line search seeks and maintains an interval (a_0, a_3) where the local optimum is supposed to be in. The size of the interval is then reduced until it is less than a predefined limit.

First three points, (a_1, a_2, a_3), are calculated in addition to the starting point a_0. Then the following algorithm reduces the interval (a_0, a_1, a_2, a_3):

```
WHILE (||a3 - a0|| > limit) DO
   IF (f(a1) > f(a2))
      a0 := a1; a1 := a2;
      a2 := a0 + 0.618*(a3 - a0);
   ELSE
      a3 := a2; a2 := a1;
      a1 := a0 + (1 - 0.681)*(a3 - a0);
   END
END
```

[2] A natural extension to this formula would be: $\mathbf{c} = \sum_{i=1}^{k} \mathbf{R}_i * \mathbf{p}_i$, where $\sum_{i=1}^{k} \| \mathbf{R}_i \| = 1$ and $\mathbf{R}_i$ are not necessarily diagonal.

This method assumes that the function to be optimized is *(pseudo)convex*, which is not always true when there are many local optima.

When it is noticed that the function within the interval may go only downhill or may not be convex the iteration stops [3]. In any case the iteration stops after `n` steps, where `n` is the dimension of the search space.

The simulation results of this algorithm are from [Aut94]

3.3 GA with Two Populations: GAG2

This algorithm is the same as GAG except it has *two populations* with the function and the gradient values, see the pseudo code stated in the beginning of this section. The population is selected before applying the crossover *box* or the *line-search* operation. However, the probabilities of choosing either one of the populations may be different depending of the particular operation ($pb1$ and $pl1$) .

When *box* has produced a child it calculates its gradient with probability $pb2$.

If the iteration of the *line-search* stops after more than one step then the gradient of the best point is always calculated. The gradient of the starting point is removed from `DP`.

If the iteration stops after one step then the gradient of the best point is calculated with probability $pl2$ unless the best point happens to be the starting point (the parent). If the best point is the starting point then the gradient of the second best point is calculated with $pl2$. [4].

3.4 Mutation and Gradient Search: MG2

The algorithm is the same as GAG2 but the *crossover* is removed, so, there is only *mutation* and *line-search*. The probabilities for applying the two operations are 0.05 and 0.95, respectively.

One may think this is not a genetic algorithm at all. It has been included to test what effect (if any) the *crossover* operator has.

4 Test Functions

The task for each algorithm is to minimize a family of modified n-dimensional Shekel's functions [Aut94]:

$$f(\mathbf{x}) = \sum_{i=1}^{m} \frac{s_i}{(\mathbf{x}-\mathbf{a}_i)^T \mathbf{H}_i (\mathbf{x}-\mathbf{a}_i) + c_i} \tag{6}$$

in a set $0 \leq x_j \leq 10$ $(j = 1, ..., n)$.

The parameter m represents the number of local optima and is uniformly distributed $(2, 20)$. The vectors $\mathbf{a}_i \in R^n$ represent the places of local optima and their components are uniformly distributed $(0, 10)$. The signs $s_i = +1, -1$ are selected at random and tell if there is a local minimum or maximum at $\mathbf{a}_i$ [5]. The scalars c_i scale the values of local optima and are uniformly distributed $(0.1, 0.6)$. Finally, $\mathbf{H}_i$s are diagonal matrices, $\mathbf{H}_i = diag(h_{1,i}, ..., h_{n,i})$, that deform the shape of the regions near local optima. They make the contours near optima elliptical. The elements $h_{j,i}$ are uniformly distributed $(0, 10)$.

Figures 1 and 2 show a two dimensional example of a modified Shekel's function and its contours (the surfaces along which the function is constant).

[3] The step size has been too big or small

[4] These rules seem to be useful. Their effects have not been systematically studied, however.

[5] Due to this there are local optima in places other than $\mathbf{a}_i$, too. So, there usually are more than m local optima.

5 Simulations

A thousand 10-dimensional modified Shekel's functions were randomly selected from the set described above. Every algorithm tried to minimize all of the functions. One run was done for each function and for each algorithm. The population size of all the experiments was 100 and the gradient population size was 50.

The length of the runs was 10000 function evaluations. The cost of one gradient evaluation is 10 function evaluations i.e. the dimension of the search space times one function evaluation.

The percentages of the global minima that a particular algorithm reached after the 1000 minimization tasks were recorded. These numbers were collected into histograms.

Figure 3 presents the results of the algorithm without gradients, GA [Aut94]. Figure 4 presents the results of the algorithm GA with golden section gradient search, GAG [Aut94]. The probabilities of mutation, crossover and line search operations were 0.05, 0.45 and 0.5, respectively.

All the simulations of the algorithm GAG2 had the same probabilities for mutation, crossover and line search as GAG above. Also in all these simulations crossover selected the gradient population with $pb1 = 0.5$ and line search calculated the gradient of the new solution with $pl2 = 0.5$, see section 3.3. Also, note the different scales of the following figures.

Figures 5 presents the result of GAG2 with $pl1 = 0.5$ and $pb2 = 0.5$. Line search selected the gradient population with the probability 0.5. Crossover calculated the gradient of the child with the probability 0.5.

The next two simulations tried to reduce the calculations of new gradients. Figure 6 presents the result of GAG2 with $pl1 = 0.5$ and $pb2 = 0.05$. Figure 7 presents the result of GAG2 with $pl1 = 0.9$ and $pb2 = 0.05$.

Figure 8 presents the results of the algorithm MG2 with two populations, mutation and line search. The probabilities of mutation and line search were 0.05 and 0.95, respectively. $pl1$ was 0.5 and $pl2$ was 0.5.

6 Concluding Remarks

Combining the golden section gradient search and the genetic algorithm produces clearly better results than the GA alone, see figures 3, 4, 5, 6 and 7. This has already been verified in [Aut94].

The basic version of GAG2 (fig. 5) performed only slightly better than the one population version GAG (fig. 4). Some parameter tuning was made to reduce the calculations of the new gradients. First, the probability of the crossover to calculate gradients was reduced from 0.5 to 0.05 (fig. 6). This made GAG2 behave better. Second, the probability of the line search to select the gradient population was increased from 0.5 to 0.9 (fig. 7). Again, GAG2 performed better. One can say that this version of GAG2 is better than GAG (fig. 4).

Finally, the version of the algorithm with no crossover, MG2, was compared against GAG2 (fig. 8). It behaved almost as well as GAG2. So, this form of crossover had not much influence on the performance of the algorithm. All these results, of course, concern only of the test functions used here.

6.1 Possible Improvements

It may be advantageous to try many gradient searches of different step sizes for one starting point. Gradient calculation is expensive, therefore trying only one search with one step size may be a "misuse" of gradient information. Then, a *population* of search intervals are got and the best ones will be taken under further consideration.

The gradient line search could be made more effective by using *interpolation* in some way. For example, given the search interval (a_1, a_2, a_3), a quadratic polynomial could be fitted to $f(a_1), f(a_2), f(a_3)$ and its minimizer chosen as the next point etc. More details of different line search strategies are found e.g. in [Fle80] pp. 25-29 and [BS79] pp. 253-268.

Instead of pure gradient line search some more delicate optimization techniques may be used e.g. Newton-like, conjugate direction, restricted step method etc. [Fle80] and [BS79]. However, it may be more useful to apply them after running some generations of the GA. So, the GA's role is to produce *starting points* to these methods that find the final local optima.

Due to gradient searches the populations may become clustered around local minima. It may be useful to check them every now and then and remove individuals that are too close to each other.

Acknowledgments: It is a pleasure to thank Dr. Kari I. Keskinen at Neste Engineering, Finland for inspiring discussions from which the idea of multicriteria populations emerged.

References

[AAK94] Jarmo T. Alander, Antti Autere, and Kari I. Keskinen. Distributed genetic algorithm for estimation of chemical reaction kinetic model parameters. In Jarmo T. Alander, editor, *The Second Finnish Workshop on Genetic Algorithms*, University of Vaasa, Palosaari, Wolffintie 36, Vaasa, Finland, March 1994.

[Aut94] Antti Autere. Combining gradient search and genetic algorithm for optimizing differentiable functions. In Jarmo T. Alander, editor, *The Second Finnish Workshop on Genetic Algorithms*, University of Vaasa, Palosaari, Wolffintie 36, Vaasa, Finland, March 1994.

[BS79] M.S. Bazaraa and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms.* Wiley, 1979.

[Coh78] Jared L. Cohon. *Multiobjective Programming and Planning.* Academic Press, 1978.

[FF93] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Stephanie Forrest, editor, *Fifth Int. Conf. on Genetic Algorithms*, University of Illinois at Urbana-Champaign, July 1993. Kaufmann.

[Fle80] R. Fletcher. *Practical Methods of Optimization Vol.1: Unconstrained Optimization.* Wiley, 1980.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley, 1989.

[LR93] Sushil J. Louis and Gregory J. E. Rawlins. Pareto optimality, ga-easiness and deception. In Stephanie Forrest, editor, *Fifth Int. Conf. on Genetic Algorithms*, University of Illinois at Urbana-Champaign, July 1993. Kaufmann.

[Sca85] J. D. Scaffer. Multiple objective optimization with vector-evaluated genetic algorithms. In J. J. Grefenstette, editor, *First Int. Conf. on Genetic Algorithms*, Carnegie-Mellon Univ., Pittsburgh, July 1985. Lawrence Erlbaum.

[SG85] J. David Scaffer and John J. Grefenstette. Multi-objective learning via genetic algorithms. In *Ninth Int. Joint Conf. on Artificial Intelligence IJCAI-85 (Vol. 1)*, Los Angeles, CA, August 1985. Kaufmann.

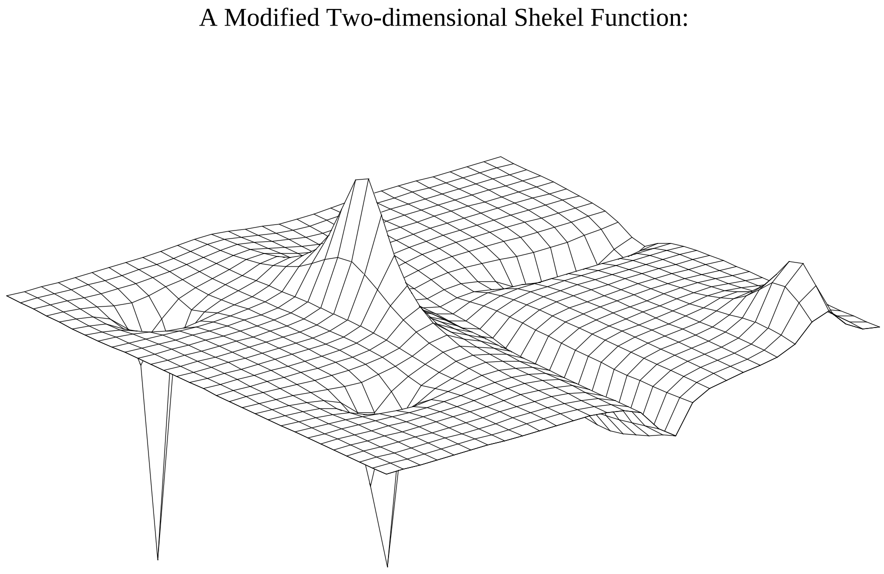

Figure 1: A modified two dimensional Shekel's function

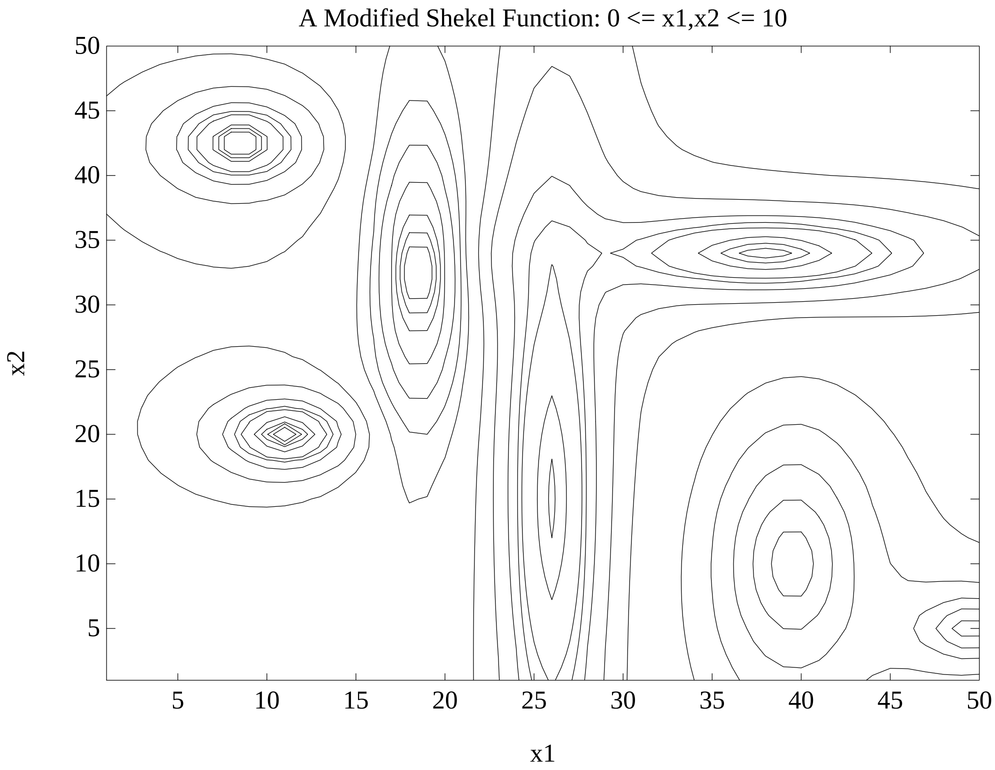

Figure 2: The contours of the modified Shekel's function

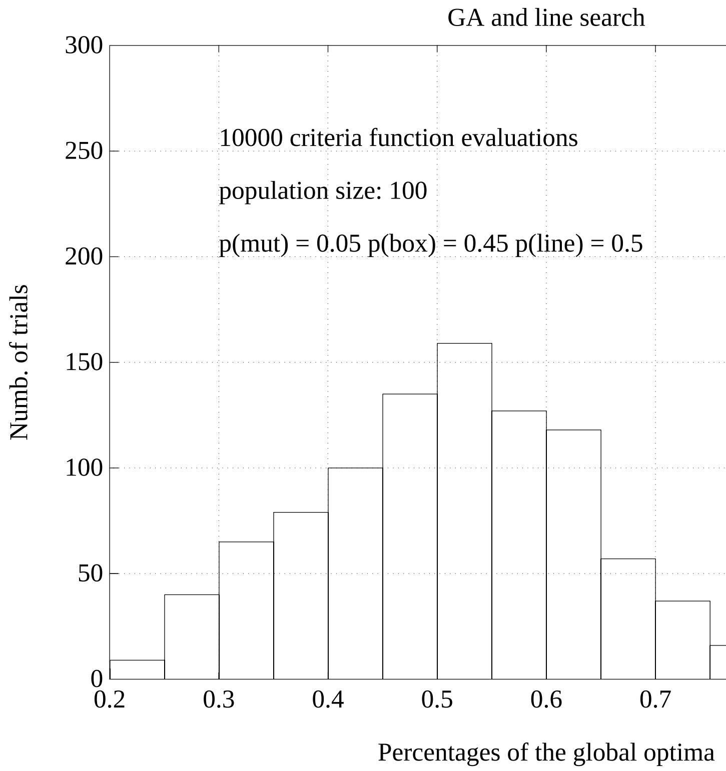

Figure 3: GA

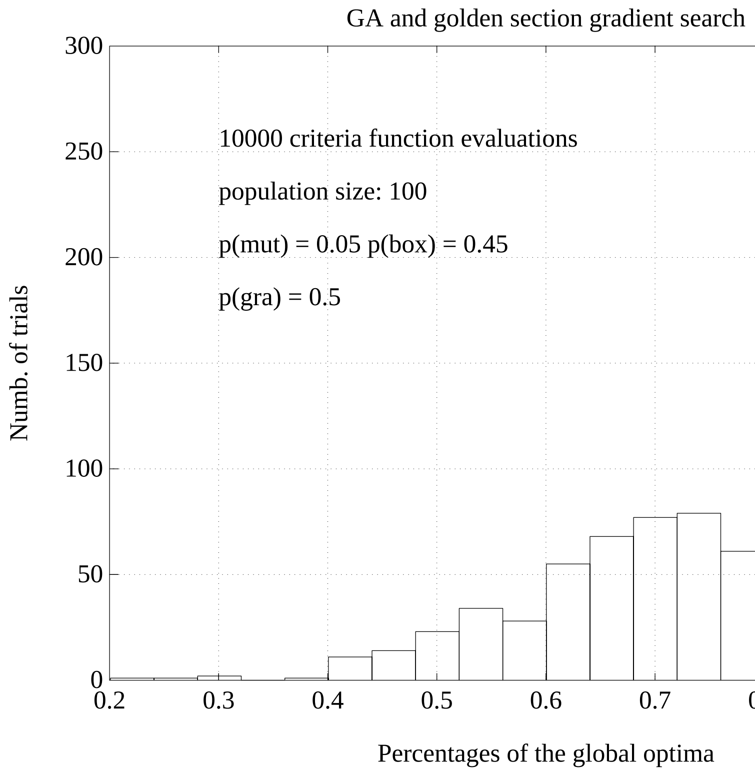

Figure 4: GAG

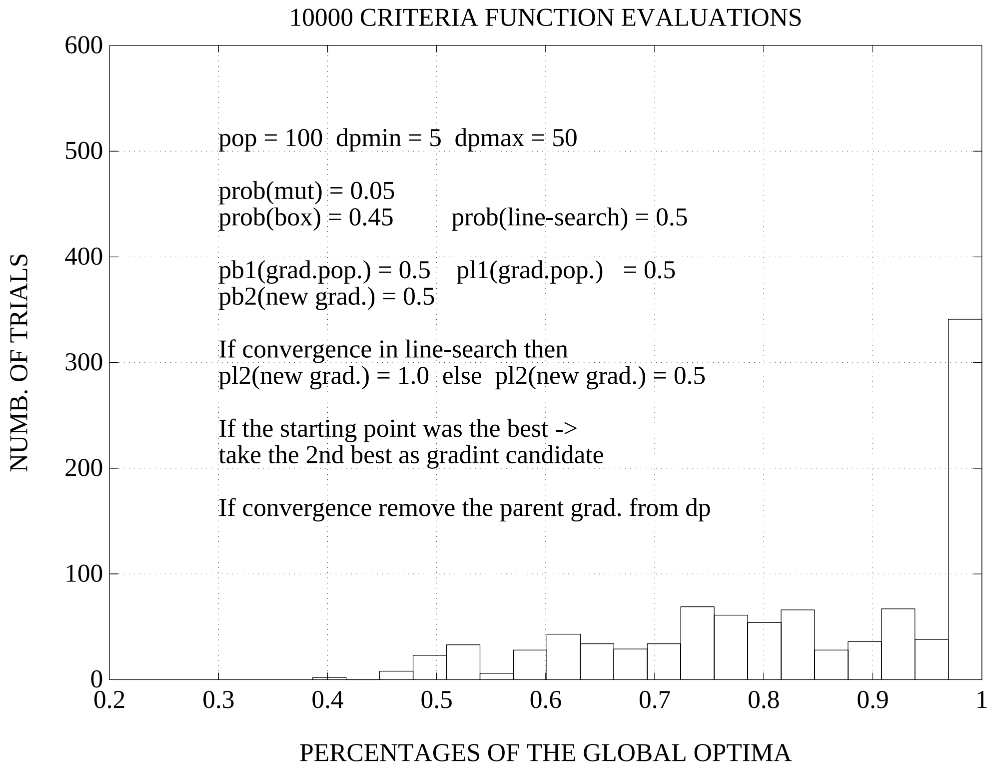

Figure 5: GAG2

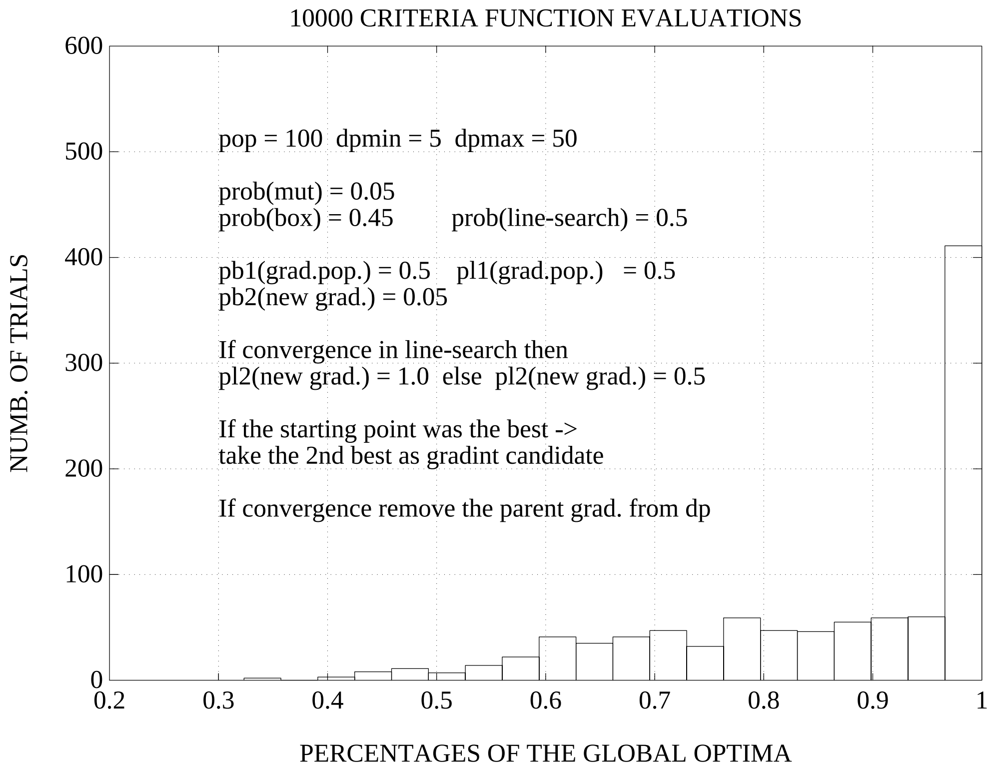

Figure 6: GAG2

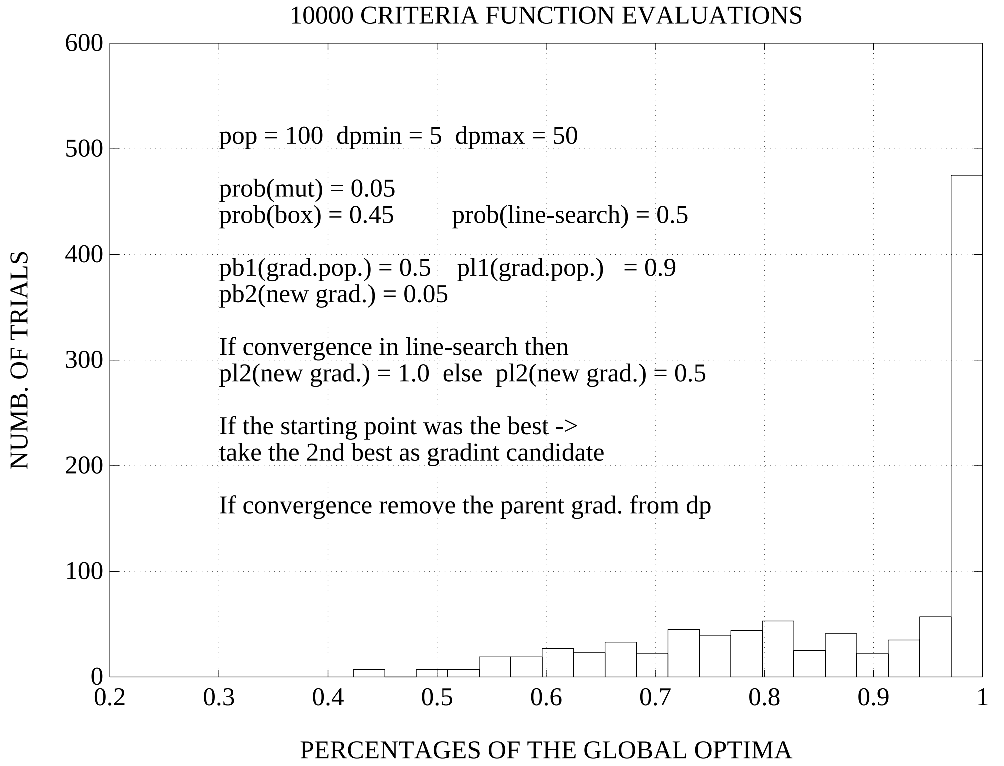

Figure 7: GAG2

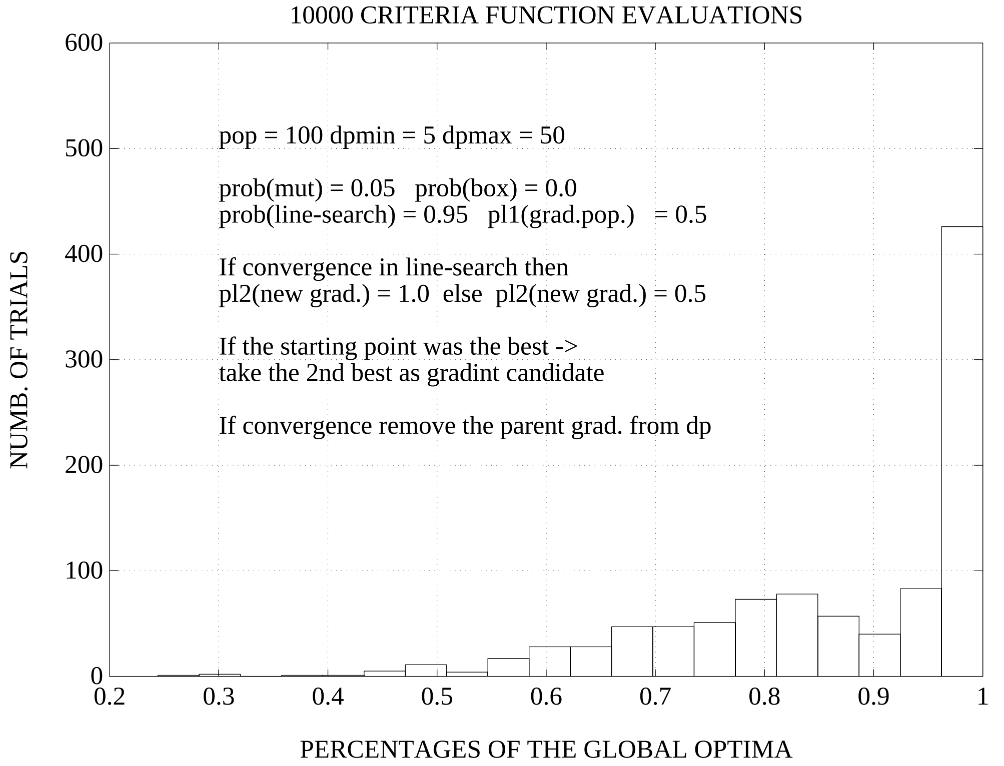

Figure 8: MG2