A Diversification Operator for Genetic Algorithms

Diptesh Ghosh

W.P. No. 2011-01-02
January 2011

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD – 380015
INDIA

A Diversification Operator for Genetic Algorithms

Diptesh Ghosh

Abstract

Conventional genetic algorithms suffer from a dependence on the initial generation used by the algorithm. In case the generation consists of solutions which are not close enough to a global optimum but some of which are close to a relatively good local optimum, the algorithm is often guided to converge to the local optimum. In this paper, we provide a method which allows a genetic algorithm to search the solution space more effectively, and increases its chance to attain a global optimum. We provide computational experience with real-valued genetic algorithms on functions of two variables.

1 Introduction

Genetic algorithms (see, e.g. Holland, 1992) are a widely applied technique in optimization problems, especially in situations where efficient algorithms are too difficult to formulate (see Karr and Freeman, 1999; Goldberg et al., 1997). These techniques are attractive since they often yield good quality solutions even when intricate details of the characteristics of the problem at hand are not clearly understood.

Genetic algorithms belong to the class of population based metaheuristics. They mimic the evolutionary process observed in the natural world, and primarily incorporate the concept of "survival of the fittest". A genetic algorithm starts out with a population (called a generation) consisting of a predetermined number of individuals. In case of optimization problems, these individuals are appropriately coded solutions to the problem. In each iteration of a genetic algorithm, the algorithm transforms the generation into the next generation by applying a certain set of operations, namely, reproduction, crossover, and mutation. The newly formed generation then becomes the starting point of the next iteration of the algorithm. The aim of the algorithm is to effectively use these operations so that each successive generation is superior to the preceding generation in terms of the objective function values of the component solutions, and the algorithm, at termination, ends up with a generation consisting of solutions of good quality, if not optimal solutions.

In a reproduction operation, a solution from a generation is simply copied into the next generation. The justification behind this operation is that a "good quality" solution in one generation is also a good quality solution in the next generation, and can aid in the process of creating the next generation. A crossover operation imitates sexual reproduction in biological evolution. In this operation, two parent solutions are chosen from the current generation based on their objective function values, and are "mated" to form child solutions for the next generation which have some desirable properties of the parents. There are two widely used methods for crossover operations. In the standard method, a common position is chosen in the coding of both parents. Then two children are produced as a result of the mating. The first child consists of the partial representation of the first parent up to the position, and the partial representation of the second parent starting from that position. The second child consists of the partial representation of the second parent up to the position, and the partial representation of the first parent starting from that position. These two children are then transferred to the new generation. In the second method (called uniform crossover), each position in the representation is randomly chosen from one of the two parents.

Experimentally, uniform crossover has been found to perform better than the standard crossover method. The mutation operation chooses solutions in a particular generation with a low probability, and with a low probability alters the value of one component of the solution chosen. This operator is not much preferred since its effect is completely random. In our paper, we do not consider the mutation operator.

Conventional genetic algorithms suffer from problems similar to local search algorithms, in that they are liable to get stuck in local optima. Consider for example, a function f_1 which is a mixture of two bivariate normal distributions, one with a mean at (1,1) and a standard deviation of 0.5 and another with a mean at (3,3) and a standard deviation of 0.25. Let us also suppose that we are interested in finding the maximum value attained by this function over the set $S = \{(x, y) : 0 \leq x, y \leq 3\}$. The shape of this function is shown in Figure 1. As is clear from the figure, f_1 has two maxima, a local maximum at (1,1) with a function value of 0.637, and a global maximum at (3,3) with a function value of 2.547.

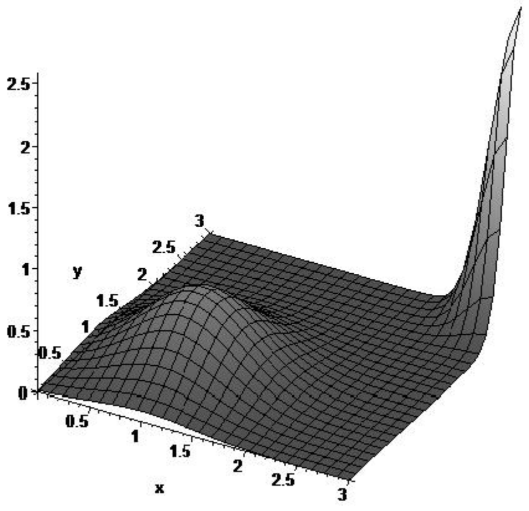

Figure 1: Function f_1

Now consider a genetic algorithm to maximize f_1 over S. Assume that this genetic algorithm encodes a solution as a vector of coordinates (x,y). Also assume that this genetic algorithm starts out with a generation of individual solutions none of which are close to the global maximum at (3,3), but several of which are close to the local maximum at (1,1). Since there are solutions close to (1,1) they are likely to have a reasonably good objective function value and hence have a high probability of being chosen for the reproduction operation. If the local maximum is a member of the initial generation of solutions, then it will certainly be chosen for reproduction. Again, since f_1 is smooth, the individuals chosen for mating in the crossover operation are very likely to be ones close to (1,1). By the very nature of the crossover operation, the children generated from such mating will again be close to the point (1,1).

So we see that in a conventional genetic algorithm, successive generations will contain solutions converging to the local optimum, ignoring the presence of a global optimum elsewhere in the domain under consideration. If a moderate quality local optimum enters a generation at any stage during the execution of the genetic algorithm, it is likely to be a member of each succeeding generation, and by its presence, will influence the solutions in each generation close to be close to itself.

Our contribution in this paper is to present a method of avoiding this premature convergence and encouraging the genetic algorithm to search the domain more thoroughly. In Section 2 of this paper, we present our methods to do this. We describe our computational experience with our methods on two complex functions in two variables in Section 3, and compare our results with those obtained through conventional genetic algorithms using the same parameters. In Section 4 we present a summary of our contribution and provide directions for future work in this area.

2 Our Genetic Algorithms

As described in the previous section, the problem with conventional genetic algorithms (referred to as `CONV_GA` in this paper) is the "immortality" of good quality locally optimal solutions, and the influence they exert to guide all solutions in subsequent generations to cluster around them. We alleviate this problem by introducing the concept of the life-span of an individual solution.

The life-span of an individual solution in a genetic algorithm is the maximum number of consecutive generations in which that solution can be propagated through the reproduction operation. Once the life-span is exceeded, the solution is not considered while designing the next generation of solutions. We include this parameter in `CONV_GA` and refer to this variation of the genetic algorithm as `EXP_GA`.

If a solution is of poor quality, then the solution will not get propagated to the next generation through reproduction, and will automatically be absent in the next generation. A good quality solution will influence a certain number of generations while its life-span is not exceeded and then be removed from future generations. Hence no individual solution will have a direct long term effect on the cohort of solutions in a particular generation. The effect of a good quality solution may however be indirect. If its quality is good, then within its life-span, it will repeatedly be chosen for mating, and will generate a relatively large number of solutions close to itself in the domain of the functions. When it is eliminated after its life-span, the other solutions which were created by it, and which, hopefully, were of good quality, will attract further generations in the vicinity of the original solution.

If the life-span of a solution in `EXP_GA` is chosen to be too small, then a good quality solution has little chance of influencing successive generations and the search becomes totally random. If the life-span is chosen to be too big, then the genetic algorithm will not be able to move away from the vicinity of a once generated good quality solution. Our next two variants of genetic algorithms aim to make genetic algorithms search in a more diversified manner by not only removing a solution from a generation once its life-span is exceeded, but by replacing it with another solution. They vary in the way the new solution is chosen.

In the `EXP_R1_GA` variation, as soon as a solutions lifespan is exceeded, it is replaced by a solution chosen randomly from the domain of the function. Since the problem is of the genetic algorithm concentrating excessively on a small part of the domain, the randomly chosen solution has a chance of being far away from the other solutions, and guiding future generations away from the previous region in the domain. In the `EXP_R2_GA` variation, as soon as a solutions lifespan is exceeded, it is replaced by a solution randomly chosen from a portion of the domain that is far away from other solutions. This choice of the new solution is exercised as follows. The domain of the function is portioned into a rectangular grid. Assume that each cell of the grid is denoted by a vector (i, j) denoting that the cell is in the i-th row and j-th column of the grid. Each solution in a particular generation lies on exactly one cell of the grid (ties are broken arbitrarily but consistently). Let n_{ij} represent the number of solutions in the current generation in cell (i, j), with the exception of the solution being replaced. Now each cell (p, q) of the grid is assigned a weight $w_{pq} = \sum_i \sum_j n_{ij}\{|p-i| + |q-j|\}$. These w_{pq} values are normalized, and the normalized values are taken as probabilities that the new solution generated will be in a particular cell of the grid. Once the cell is determined, a solution is chosen randomly from the part of the function domain represented by that cell.

3 Computational Experience

We studied the performance of all four variations of genetic algorithms on two functions of two variables. The four variations were coded both with the standard crossover operator and the uniform crossover operator. For all our programs, we took the size of each generation to be 25, and stopped after the genetic algorithm had created 50 generations. We stipulated that the best five solutions in each generation will be reproduced in the next generation. For the crossover operation, we chose the parent solutions in such a way that a solution with high objective function value has a high chance of being selected to mate. Since the functions tested were of two variables, the standard crossover operation was trivial; the first child solution consisted of the x-coordinate of the first parent and the y-coordinate of the second parent, while the second solution consisted of the x-coordinate of the second parent and the y-coordinate of the first parent. In the uniform crossover operation, the values of the x- and y-coordinates of the first parent have a 60% chance of being copied to the corresponding coordinate in the child, while the values of the x- and y-coordinates of the second parent have a 40% chance. The initial generation of solutions was deliberately chosen to avoid the neighbourhood of the global maximum. In addition, a local maximum was chosen as one of the solutions in the initial generation. The life-span of any solution for `EXP_GA`, `EXP_R1_GA` and `EXP_R2_GA` was chosen to be two iterations.

The metrics that we used to compare the variations are (1) the best objective function value obtained by the algorithm at each generation; (2) the diameter of each generation of solutions; and (3) the thoroughness with which the genetic algorithm has explored the domain of the function. For this purpose the diameter of a generation of solutions is defined as the maximum Euclidean distance between two solutions in the generation. A low value of diameter is an indicator that the solutions in a generation are tightly clustered in the domain of the function being maximized. A high value is not an indicator of high diversity; all but one solution may be clustered together and one solution may be far away in a generation with high diameter. The thoroughness of exploration is measured by first partitioning the domain using a grid, counting the number of solutions that the genetic algorithm generated in each cell of the grid, normalizing these values and finding the standard deviation of the normalized values. If the number of solutions generated by two genetic algorithms is the same, then a lower value of standard deviation indicates that a particular genetic algorithm has searched the solution domain more thoroughly.

We now compare the performance of the four variations on f_1 over the domain $S = \{(x, y) : 0 \leq x, y \leq 3\}$. In this comparison, we ensure that all four variations start with the same initial generation of solutions, and choose the standard crossover operation. As mentioned in the introductory section, none of our variations includes a mutation operation. Figures 2 through 4 depict the results of the comparison. Recall that the minimum value of f_1 over S is very close to 0.0 and the maximum value is 2.547 at the point (3,3). The best solution in the initial generation is (1,1) with an objective function value of 0.637. Notice from Figure 2 that `CONV_GA` and `EXP_GA` did not improve this solution over the 50 iterations which both the algorithms were allowed. On the other hand, both `EXP_R1_GA` and `EXP_R2_GA` managed to improve upon this solution during the execution of the algorithm. So at the end of execution of these four variations, the best solution obtained by `EXP_R1_GA` and `EXP_R2_GA` are better than those obtained by the other two variants. The best solution obtained by `EXP_R1_GA` is (2.807,2.905) with an objective of 1.760 while the best solution obtained by `EXP_R2_GA` is (2.899,2.892) with an objective function value of 2.138. In other words, both of these variations could move out of the vicinity of the local maximum at (1,1) and evaluate solutions near the global maximum at (3,3).

Next let us compare the diameters of the generations produced by the four variants. The variation of these diameters with the index of the generation is shown in Figurefig:dia. Notice that the diameters for `CONV_GA` and `EXP_GA` reduce drastically with the generation number, and within 15 generations, reach a value of 0. This means that running these two variations after 15 iterations does not change the objective function value of the best solution obtained by these variation. This

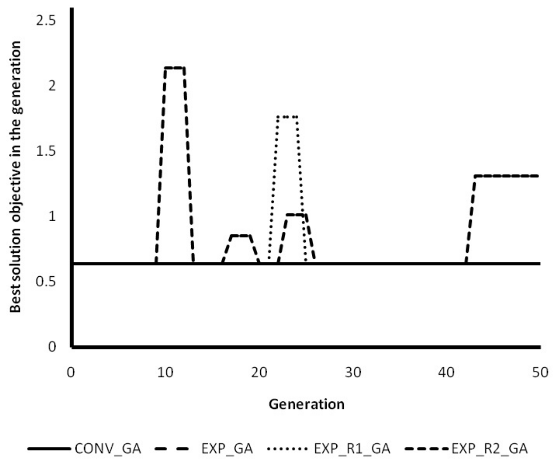

Figure 2: Objective function values of the best solution in each generation for genetic algorithms using standard crossover operation on function f_1

is logical, since in both these variations, there is no mechanism for diversification of solution generations. In case of the other two variations however, the diameter of the generations do not follow a monotonic trend. In both the variations, they show sudden large increases.

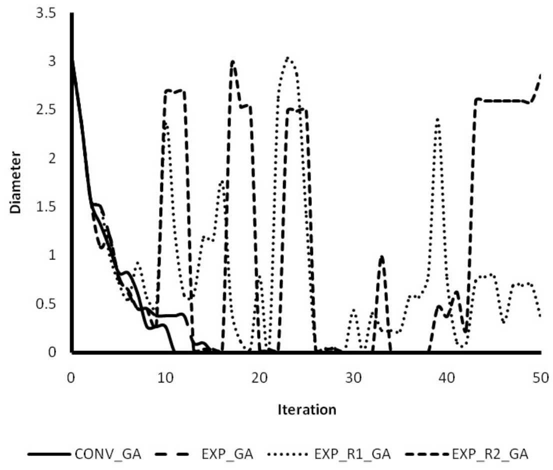

Figure 3: The diameter of each generation for genetic algorithms using standard crossover operation on function f_1

Comparing with Figure 2, we observe that improvements in the objective function value of the best solution found by the variations closely follow some of these increases. This supports the fact that when the life-span of a particular solution ends, and a new random solution is generated in its place, occasionally, the new solution is present at a point in the domain close to the global maximum and hence has high objective function values.

Finally we compare the thoroughness of exploration of the solution domain by the four variants. Since each of the variants produce 25 solutions in each generation and each runs for 50 generations, the total number of solutions produced by each of the variants during execution is identical, i.e., $25 \times 50 = 1250$. Figure 4 shows the counts of the number of solutions in each cell of a 5×5 grid covering the function domain. Cell (2,2) corresponds to the region in the domain close to the local maximum at (1,1), while cell (5,5) corresponds to the region closest to the global maximum at (3,3).

3	3	4	2	0
2	1150	44	6	2
5	28	14	2	2
0	1	0	0	0
1	1	2	2	1

CONV_GA

3	2	5	2	0
1	1121	42	6	2
5	45	29	2	2
0	1	0	0	0
1	1	2	2	1

EXP_GA

3	13	4	2	0
5	1043	80	10	12
5	48	23	2	2
0	3	0	0	0
1	11	2	2	4

EXP_R1_GA

3	7	3	2	0
1	973	59	7	32
4	30	15	2	3
0	1	0	0	0
1	30	5	2	95

EXP_R2_GA

Figure 4: Distribution of solutions generated by the four variants over the domain S of the function f_1

The standard deviations of the normalized count scores are 0.180 for CONV_GA, 0.175 for EXP_GA, 0.163 for EXP_R1_GA, and 0.152 for EXP_R2_GA. This indicates that EXP_R2_GA searched the function domain most comprehensively, while the search by CONV_GA was localized. This is in agreement with the fact that EXP_R2_GA produced the best solution at the end of its execution.

When we performed the same analysis with the four variations on the same problem, but using uniform crossover instead of the standard crossover operation, we observed similar results. Figure 5 shows the variation of the objective function values of the best solution in different generations by the four variations, and Figure 6 shows the variation in the diameters of the different generations of solutions. The standard deviations of the normalized count scores are 0.187 for CONV_GA, 0.185 for

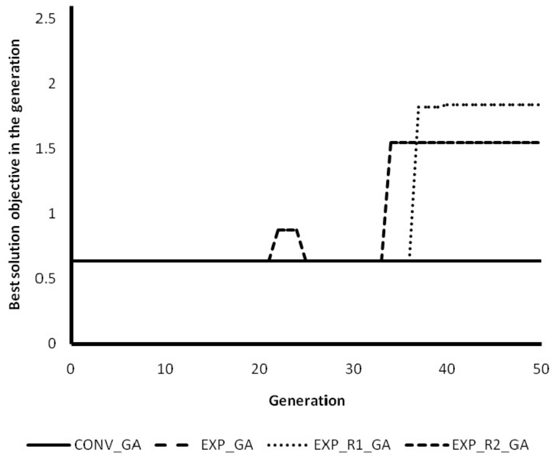

Figure 5: Objective function values of the best solution in each generation for genetic algorithms using uniform crossover operation on function f_1

EXP_GA, 0.141 for EXP_R1_GA, and 0.136 for EXP_R2_GA. Note that here too, CONV_GAand EXP_GAperform poorly, and never show an increase in the diameter of a generation, while EXP_R1_GAand EXP_R2_GA perform much better. Note also, that in this problem, the standard crossover operation generates a better solution than uniform crossover operation.

We ran the four variations of genetic algorithms from ten different initial generations of solutions. Table 1 presents a summary of our findings from the ten runs using the standard crossover operator. The second column of the table shows the average of the objective function values of the best solution

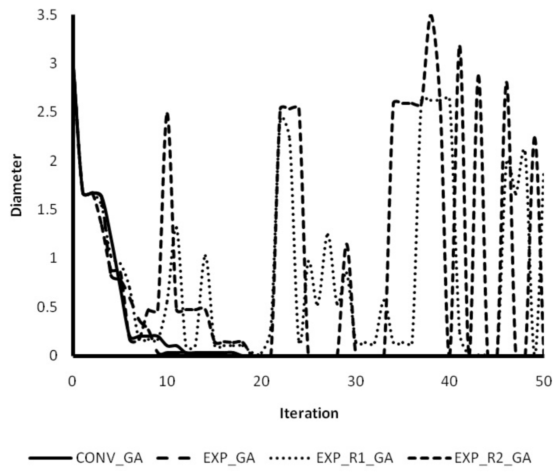

Figure 6: The diameter of each generation for genetic algorithms using uniform crossover operation on function f_1

obtained in all ten runs. The third column shows the average number of iterations required to first reach the best solution in a particular run. The fourth column shows the number of runs in which the best solution obtained by the genetic algorithm is in the vicinity of the global maximum.

Table 1: Performance of genetic algorithms with standard crossover on f_1

Algorithm	Obj. fn. value of best soln.	Itn. to reach best soln.	Times GA reaches near best soln
`CONV_GA`	0.637	1.0	0
`EXP_GA`	0.637	1.0	0
`EXP_R1_GA`	1.578	26.1	9
`EXP_R2_GA`	1.865	16.1	9

From Table 1 we see that both `CONV_GA` and `EXP_GA` were never able to improve upon the best solution seen in the initial generation. Recall that by design this was a locally optimal solution. This is understandable, since neither `CONV_GA` nor `EXP_GA` contained any mechanism for diversification. `EXP_R1_GA` and `EXP_R2_GA` could move away from this solution in a number of runs, in fact, `EXP_R2_GA` was always able to move away from the local maximum. The average quality of solutions obtained by `EXP_R2_GA` was better than that obtained by `EXP_R1_GA`. Table 2 presents a summary of our findings from the ten runs using the uniform crossover operator. The structure of Table 2 is identical with that of Table 1.

Table 2: Performance of genetic algorithms with uniform crossover on f_1

Algorithm	Obj. fn. value of best soln.	Itn. to reach best soln.	Times GA reaches near best soln
`CONV_GA`	0.637	1.0	0
`EXP_GA`	0.637	1.0	0
`EXP_R1_GA`	1.478	22.0	7
`EXP_R2_GA`	1.350	13.9	10

From Table 2 we see trends similar to that seen in Table 1. The only difference is that for this crossover operator, `EXP_R1_GA` performs marginally better than `EXP_R2_GA` in terms of average solution quality. Another feature to notice is that the standard crossover operator seems to produce better quality solutions than the uniform crossover operator, especially for `EXP_R2_GA`.

Apart from f_1 described in the introductory section, we performed the same experiment with another function, which we call f_2. The domain of this function was chosen as $S_2 = \{(x, y) : 0 \leq x, y \leq 10\}$. This function is a mixture of eight bivariate normal distributions, centred around (3,3), (3,7), (7,3), (7,7), (7,9), (8,8), (9,7) and (9,9). By altering the standard deviations of the normal distributions, we chose (8,8) to be the global maximum with an objective function value of 1.041, and the four neighbouring local maxima to be of moderate quality, having objective function values of 0.639 each. The other three local maxima had objective function values of 0.249 for the maximum at (3,3) and 0.159 for the maxima at (3,7) and (7,3). Figure 7 shows the plot of f_2 over S_2.

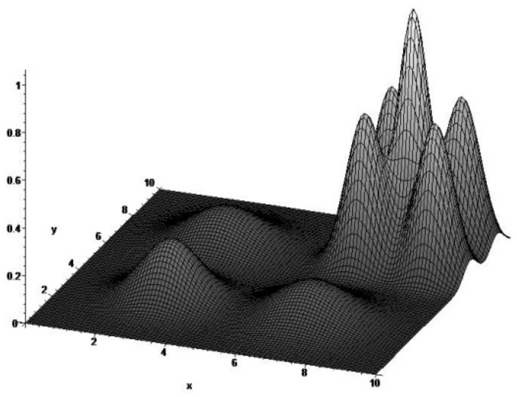

Figure 7: Function f_2

As in case of f_1, we chose ten initial generations of solutions which were far from the global maximum at (8,8). The initial generations did not have any solutions in which both the x and y coordinates were between 6 and 10. We added a locally optimal solution (3,3) to the initial generation. We fixed the number of solutions in each generation at 25 and allowed the genetic algorithms to run for 50 generations. The life-span of solutions for `EXP_GA`, `EXP_R1_GA` and `EXP_R2_GA` was chosen to be two iterations. Table 3 summarizes our experience with the four variations over the ten runs using the standard crossover operator and Table 4 summarizes our experience with the uniform crossover operator. The structures of the tables are identical with the structure of Table 1.

Table 3: Performance of genetic algorithms with standard crossover on f_2

Algorithm	Obj. fn. value of best soln.	Itn. to reach best soln.	Times GA reaches near best soln
`CONV_GA`	0.336	1.4	0
`EXP_GA`	0.336	1.4	0
`EXP_R1_GA`	0.790	26.5	8
`EXP_R2_GA`	0.687	25.2	4

It is clear that the main inferences from these tables are similar to the inferences from Tables 1 and 2. We see that neither `CONV_GA` nor `EXP_GA` can get out of the local maxima reached in the initial generation. `EXP_R1_GA` and `EXP_R2_GA` can both get out of the local maximum. In this problem, clearly `EXP_R1_GA` outperforms `EXP_R2_GA`, both in terms of the average quality of the best

Table 4: Performance of genetic algorithms with uniform crossover on f_2

Algorithm	Obj. fn. value of best soln.	Itn. to reach best soln.	Times GA reaches near best soln
CONV_GA	0.307	1.6	0
EXP_GA	0.311	1.5	0
EXP_R1_GA	0.813	24.4	8
EXP_R2_GA	0.695	21.4	2

solution encountered by the algorithm, and the frequency with which the best solution is in the neighbourhood of the global maximum. Here the difference between the two crossover operators is not observed to be significant.

4 Summary

In this paper, we have pointed out a weakness in conventional genetic algorithms. We argued that if a genetic algorithm uses reproduction and crossover operators for its functioning, then successive generations constructed by the algorithms have a chance of getting concentrated around local optima, without giving the genetic algorithm a possibility of looking at points in the domain which are far away from that local optimum but close to a global optimum. We also argued that if a local optimum of sufficiently high quality is encounter "immortal", i.e., present in all successive generations, and its solution quality will ensure that it pulls individuals towards itself in successive generations, causing the genetic algorithm to converge prematurely to the local optimum. We proposed two diversification mechanisms in this paper which prevents such convergence. We tested the diversification ideas introduced here on two functions of two variables. Our computational experiments validated our arguments on the drawbacks of conventional genetic algorithms, and showed that genetic algorithms with diversification processes built in perform much better in the search for a globally optimal solution.

There are several ways in which the idea introduced in this paper can be studied more deeply. We mention two immediate extensions here.

The diversification process introduced here necessarily requires a solution to be eliminated after its life-span is over. In our experiments we have maintained the life-span as a constant of two iterations. It is interesting to check the effect of (a) different values of life-spans, and (b) life-spans varying with solution properties on the quality of solutions generated by genetic algorithms.

All experiments in this paper have been carried out on continuous functions of two variables. In the literature, realvalued genetic algorithms have been applied on functions of higher dimensions (see Digalakis and Margaritis, 2002). The other main application of genetic algorithms, namely in combinatorial optimization problems encounters non-smooth objective functions in many dimensions. It will be interesting to see how the algorithms described here scale up when the dimension of the problem increases. We conjecture for example, that EXP_R1_GA will mostly be unaffected, but EXP_R2_GA will have to be designed carefully when dealing with problems of higher dimensions.

References

J.H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd edition, MIT Press, 1992.

C.L. Karr and L.M. Freeman, *Industrial Applications of Genetic Algorithms*, CRC Press, 1999.

D.E. Goldberg, K. Zakrzewski, B. Sutton, R. Gradient, C. Chang, P. Gallego, B. Miller, and E. Cantu-Paz, Genetic algorithms: A bibliography, IlliGAL Report No. 97011, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1997.

J.G. Digalakis and K.G. Margaritis, An experimental study for benchmarking functions for genetic algorithms, *International Journal of Computer Mathematics, 79,* 403–416, 2002.