Mixing Paradigms: A Genetic Operator that Approximates Gradient-Descent

MEVAN RANASINGHE
Stanford Electrical Engineering Department
Stanford University
Stanford, California 94305
mevanr@stanford.edu

ABSTRACT

A new genetic mutation operator was created to approximate gradient-descent techniques in a genetic algorithm. The performance of this genetic operator was evaluated on the search space of the five De Jong test functions. The results of GA runs with this Gradient-Descent Mutation (GDM) operator were compared to the same GA with the standard mutation operator – the results were greatly in favor of the GDM operator with as many as 4 of 5 runs producing a fitter individual with this new technique. The possibilities for enhancement of this specific GDM operator are discussed along with general possibilities of this hybrid paradigm.

1. Statement and Motivation of the Problem

While an evolutionary search is a robust approach to seek solutions in a search space, other more mainstream paradigms offer specific advantages. In this paper, I implement a genetic algorithm that features a new mutation operator that approximately recreates a gradient-descent technique. While a purely calculus-based gradient-descent technique may converge to a sub-optimal solution in a non-linear search-space (in local minima or maxima), a hybrid of both techniques pose an interesting situation. Could we effectively combine the efficiency of hill-climbing gradient-descent in a more general robust genetic operation?

First, we must realize that hill-climbing is most effective in linear regions. The drawback is that most non-trivial search spaces are non-linear, and a robust solution such as a genetic search usually outperforms pure gradient descent techniques (Goldberg 1989) in these real-world search spaces. The motivation for this paper lies in the fact that many non-linear search spaces can be approximated to be linear if we consider a much smaller sub-space of the whole search space. So if a genetic operator can evolve a solution to the neighborhood of an optimal solution, it is easy to see how a gradient descent technique could be very effective in finishing the search process that genetic operations may have initiated.

The aim is to create a Genetic Algorithm that exploits linear regions in a search space – just as Genetic Programming seeks to exploit repetition through the creation of ADFs for reusing code (Koza 1992). While many search problems may not be conducive to such exploitation of local linear regions in a search space, the hope is that whenever they are present, a hybridized GA approach will exploit linear regions (like ADFs exploit repetitious patterns in a GP search space). The ultimate aim is to converge on better solutions, faster.

It is necessary to note that the approximation of gradient descent does not apply to all GA problems. We can discuss the concept of gradient for problems where the gene represents a granulated (or discretized) value of a continuous variable, but NOT for discrete variables or Boolean variables. For example, it may be useful for the 10-member truss problem (Goldberg 1989) where each gene represents a vector-valued dimension of a truss thickness granulated over a continuous range, but not for the Hamburger Optimization problem (Goldberg 1989). There is no practical reason why it cannot be used for these discrete variables – it just does not represent gradient descent because we cannot talk of gradient in a non-numerical representation of the gene such as the decision whether to have fast service or slow-leisurely service for the hamburger problem which represents a series of independent choices where the concept of gradient does not apply. However, having mentioned this distinction, it is necessary to note that this very distinction is lies in the interpretation of the gene to the real-world problem, not the structure of the gene itself, and thus there is no reason why gradient descent would not work – although we would not expect it to produce faster convergence to a solution. In the context of this paper, let us consider a gene that represents a discretized value of a continuous variable and explore a gradient-descent implementation in a GA.

2. Major Preparatory Steps

Since the major focus is to implement the idea stated above in a GA, the first step is to create a 'genetic' operator that closely replicates a gradient descent technique. Although there may be many methods for creating such an operator, the one evaluated in this paper was conceived as an extension of the standard mutation operator.

2.1 Gradient-Descent Mutation

The Gradient-Descent Mutation operator (GDM) starts by mutating a random bit position in an individual. However, unlike standard GA mutation it has an extra step - if the first mutation produces a fitter individual than its pre-mutation parent, GDM immediately executes a follow-up mutation in an adjacent bit position. This follow-up mutation is not to a random bit value. Instead, the value that the new bit takes during this follow up mutation is the same bit value as the original mutation. This is repeated for another follow up mutation to create a series of mutations until either a GDM follow-up mutation produces an individual that is not as fit as its parent or a maximum number of series mutations are reached.

Table 1 – GDM steps illustrated on a 6-bit binary string

Evolution of Individual	String	Fitness	Next GDM Decision
Original individual	`010010`	0.85	Random selection of mutation at position 4
Random-mutation result	010**1**10	0.75	0.75 < 0.85 induces follow-up mutation at position 3
First Follow-up mutation	01**1**110	0.58	0.58 < 0.75 induces follow-up mutation at position 2, but since it is already equal to 1, mutation at pos 2
Second follow-up mutation	**1**11110	0.68	0.68 > 0.58 halts GDM

The example above (Table 1) on a random 6-bit string illustrates GDM. Note that this example is specific to mutation done to left bit positions from the original bit position. We shall take as an example a randomly selected individual `010010`. In GDM the first mutation is randomly selected at position 4 where the original 0 bit is changed to the value 1. Since this produces a fitter individual than the original, a follow-up mutation is done at the left adjacent bit position (position 3), which is also changed to a value of 1. If this produces an individual fitter that its parent (NOT the original individual) we do another follow-up mutation at position 2. Since this bit is already a 1, we move to the next left most adjacent position, which is then changed to a 1. However, in this instance the fitness of the offspring is not as desirable as its parent, and thus the GDM operation is terminated for this individual.

The flowchart overleaf (Figure 1) summarizes the operation in comparison to a standard mutation operator (Goldberg 1989) implemented in Genesis (Grefenstette 1990). The flowcharts below are for generalized procedures – neither the maximum number of times the follow-up mutation takes place nor the nature of the adjacent position (left or right) is specified. Unpacking (and packing) refers to the process of condensing the individual chromosomes and is merely pre-processing (and post-processing) that is done by algorithms such as Genesis for greater efficiency.

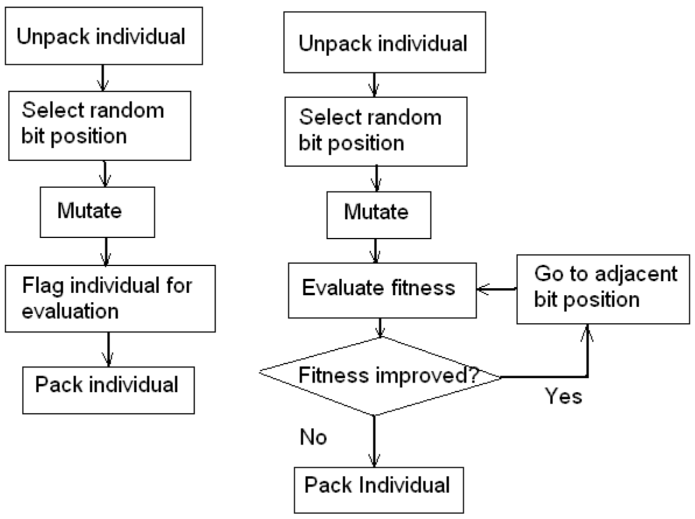


Figure 1 – Flowchart comparison of a standard GA mutation operator (left) to GDM (right)

How does this operation approximate gradient descent? Qualitatively, gradient-descent increments the independent variable in the direction of steepest gradient in order to converge on a local minimum. The GDM operator works in the same manner. First we must understand that mutating a bit to 1 will increment the total vector value of the gene, while mutating the bit to 0 decrements the vector value. If an incremental mutation (to value 1) creates a fitter offspring, gradient descent would continue to increment the independent variable (mutate more values of the gene to 1) to continue to make the next generation of offspring even fitter. Likewise, if decrementing the value of the gene (with a mutation of 0) creates a fitter individual, then consecutive mutations would attempt to decrement the value further (by more bit mutations to a value of 0). It must be noted that adjacent bit positions have different contributions to the final vector value of the gene. Thus, a follow-up mutation at a left-adjacent position will have twice the effect in incrementing the vector value of the gene than the original mutation. Similarly, the follow-up mutation of a right-adjacent position will have half the original increment. Thus, GDM is an exponentially incrementing gradient-descent approximation as opposed to traditional techniques where the increment is constant.

2.2 De Jong Test Functions

The five De Jong test functions were original proposed (DeJong 1975, Goldberg 1989) as a means of measuring the abilities of search algorithms. These are used to compare the effectiveness of the GA empowered by GDM to a GA with standard genetic operators. They are shown below (Table 2). Note that the implementation of Test Function 5 (denoted *F5** in Table 2) have the following characteristics: (i) The constants a_{ij} are stipulated as i+j (ii) Large values of the function (corresponding to the zones very near asymptotes) are given a maximum allowable value (1000) in order for the code to be executable (iii) the constant 0.002 was changed to $2x10^{-8}$ & (iv) the output was scaled by a constant (10^7) so that results were apparent within the 4 significant digit outputs of Genesis. The last change two changes were a necessity in order to differentiate between the fittest individuals. The differences between relatively fit individuals were often in the order of magnitude 10^{-8} due to the asymptotic nature of the function. Modifying the Genesis code to display results in at least 8 significant digits would make these last two modifications unnecessary. However, this was found to not be easily achievable. Scaling the search space by a factor of 10^7 does not change any of the characteristics of this complicated search space but merely allows differences between individuals to be amplified so that the output produced by Genesis displays individual differences. The search spaces below are not large by today's standards but they are sufficient to illustrate the relative effectiveness of GDM.

Table 2 – De Jong Test functions and their corresponding search spaces

	Function	Genes	Range for each gene	Structure Length
F1	$\text{Sum}_i(x_i^2)$	i=3	$-5.12 < x_i < 5.12$	30
F2	$100(x_1^2 – x_2) + (1- x_1)^2$	i=2	$-2.048 < x_i < 2.048$	24
F3	$\text{Sum}_i (\text{integer} (x_i))$	i=5	$-5.12 < x_i < 5.12$	50
F4	$\text{Sum}_i (i\ x_i^4) + \text{Gauss}(0,1)$	i=30	$-1.28 < x_i < 1.28$	240
*F5**	$0.002 + \text{Sum}_j (1/[\text{Sum}_i (x_i - a_{ij})^6])$ j = 1 to 25	i=2	$-65.536 < x_i < 65.536$	34

3. Replication of Results

3.1 General Parameter values

In order to create a fair tableau for comparison of the standard GA with a GDM GA, the population size and number of total evaluations was kept the same – a modest population of 30 and 10 generations.

The standard GA was run with the parameters recommended by Goldberg (Goldberg 1989) shown below (Table 3). The new GDM GA was run at two separate parameter sets as shown below. This tableau was used for all five test functions. GDM run 1 and 2 only differ by the crossover and mutation rates. The values used here were chosen as a result of prior experimentation that showed that a large proportion of GDM is necessary in order for the GDM GA to produce significant positive results.

Table 3 – Tableau for each GA run

Parameter	**Standard GA**	**GDM – run 1**	**GDM – run 2**
Experiments	1	1	1
Population	30	30	30
Generations	10	10	10
Crossover Rate	0.6	0.5	0.3
Mutation Rate	0.033	0.5	0.7
Random seed	1234567	1234567	1234567
Fitness Function	See Table 2	See Table 2	See Table 2
Structure Length	See Table 2	See Table 2	See Table 2
Wrapper	None	None	None

3.2 GDM Parameters

Since GDM itself has several options in terms of its implementation, it is necessary to denote the parameter choices used in this paper. First, the maximum number of consecutive mutations was set at 5. This choice was arbitrary - we need a significant number of follow-up mutations relative to the gene length to ensure that it has a palpable impact while at the same time not too long as for it to dominate the entire gene with the same bit value.

Next, the follow-up mutations were all done to consecutive bits on the *left* side of the original random bit position. This could be easily modified to provide mutations to the *right* of the position, or in fact to both sides of the random initial position.

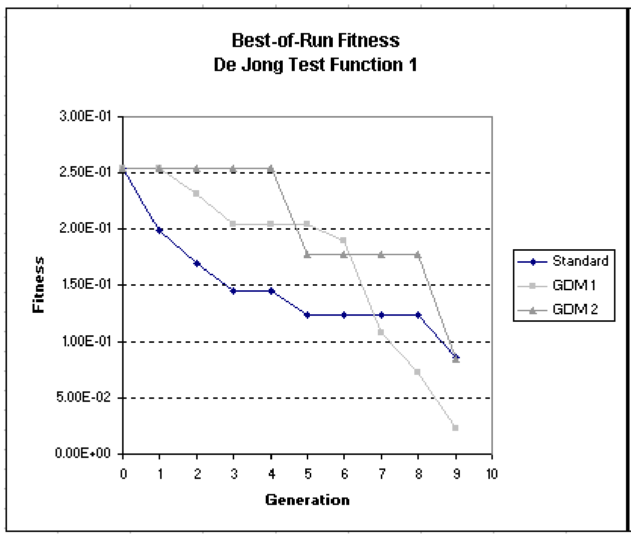
4. Results and Observations

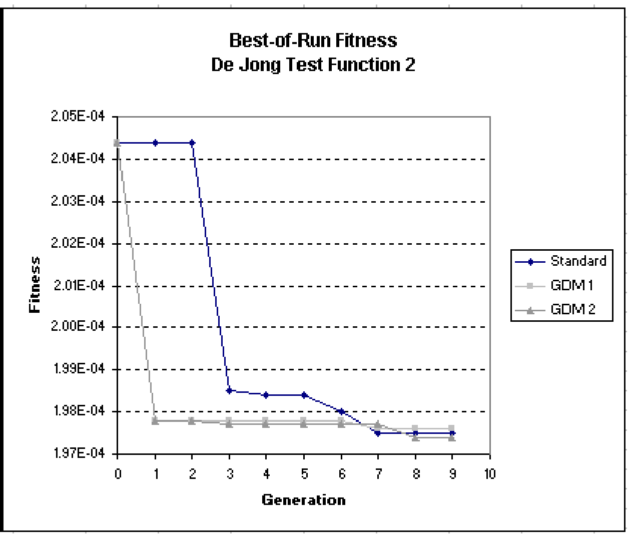
The fitness values of the best-of-run individuals for each run are shown below (Table 4). Only one run was performed for one pre-selected random seed due to the limited time. However, verification of the results should be carried out with multiple runs and hypothesis testing as described in the proceeding section 7.

Also note that the basis for evaluation of performance, namely the computational effort, has been organized as the total number of trials and thus the best-of-run individual fitness is compared after the 10 generations are completed.

Table 4 – Best-of-run individual fitness values

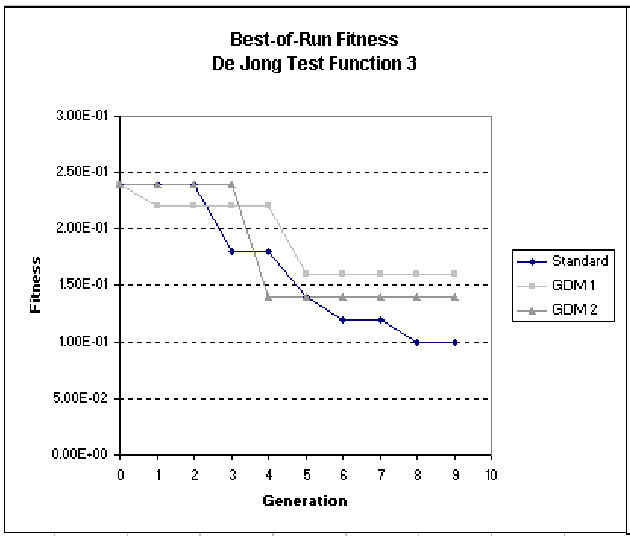
Function	Standard	GDM 1	GDM 2
F1	8.58E-02	2.33E-02	8.43E-02
F2	1.98E-04	1.98E-04	1.97E-04
F3	1.00E-01	1.60E-01	1.40E-01
F4	5.19E-01	3.80E-01	5.06E-01
F5*	4.94E-01	4.46E-01	4.45E-01

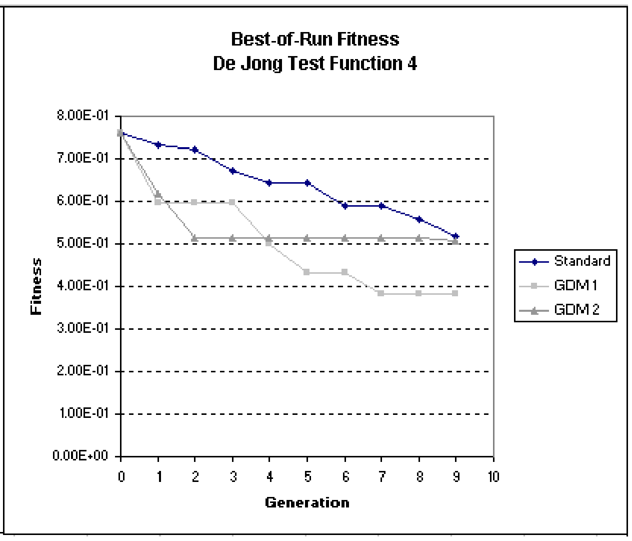




Figures 2,3 –Graphs of results from DeJong test functions 1(left) and 2 (right)

In Figure 2, both GDM runs produced better results than the standard GA – especially GDM 1, which was almost 4 times better than the standard GA result. In Figure 3, all the runs produced results that were very similar.





Figures 4,5 –Graphs of results from DeJong test functions 3 (left) and 4 (right)

Figure 4 shows the only test function where the standard GA produced better results than the GDM runs. Figure 5 shows the expected success of both GDM runs producing better results.

Figure 6 below again shows that both GDM runs were similar to each other – both outperforming the standard GA.

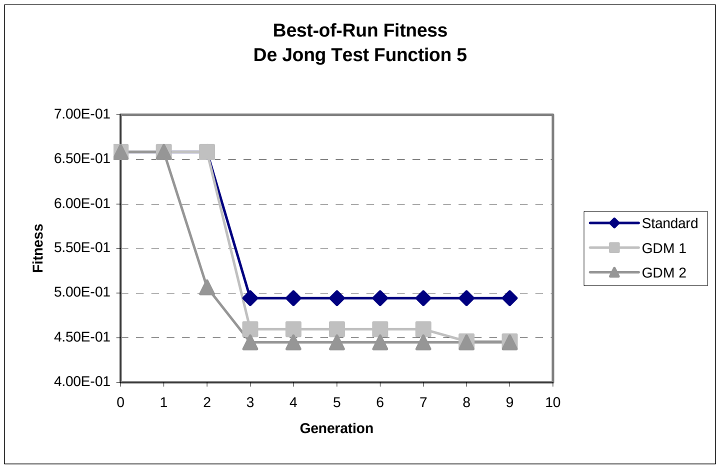


Figure 6 –Graph of results from DeJong test function 5

5. Discussion and Interpretation

First, we can see that GDM produced better results for 4 of the 5 test functions. The counterargument as to the validity of having 2 runs in GDM versus the 1 run of the standard GDM can be rebutted by the following statement: the overall *worst* best-of-run individual of the three runs was seen with the standard GA (test functions 1,4,5). The standard GA produced approximately equivalent results in one search space (test function 2) while it was clearly better in the remaining case (test function 3).

Next, in order for these exploratory results to carry any significant data-driven authority, they need to be repeated for many runs and possibly in other search spaces. The GA runs in this paper, by themselves, are insufficient in number to be used to assert the overall out-performance of GDM. However, it does show the potential ability of such gradient-descent operators.

Finally, the results indicate that GDM does not function in a similar manner to the standard mutation operator. Where the latter tends to introduce genetic diversity to the population, the former tends to produce more homogenous individuals. Particularly, it can be reasoned that schemata where consecutive bits have the same value are propagated. This immediately brings into focus the necessity of standard mutation when GDM is used to prevent premature sub-optimal convergence of the population. Thus, like crossover, GDM tends to produce fitter individuals. The next step in the exploration of the ability of GDM would be to combine standard mutation, crossover *and* GDM in the same GA run.

A special note as to the nature of the comparison between the GA runs should be made at this point. Each follow-up mutation in GDM requires an evaluation that is not totaled into the final number of trials. This constitutes an unfair advantage that needs to be compensated in order to form a better comparison of the performance of this operator.

6. Conclusion

This project was a success in illustrating the potential of an exceedingly simple gradient descent approximation. An extensive search in the literature is necessary to discover any work that is inspired by similar ideas. The coding of GDM in C was carried out for effect and not efficiency – thus it can be made more efficient in order to make the evaluation faster.

It is worth noting that the success of gradient descent in a genetic operator poses a question in terms of its possible interpretations. Could gradient-descent represent a migratory behavior of genes towards an optimal configuration? Could this possibly occur in nature? Can this idea be extended into biological genetics?

The genetic paradigm is a relevant one in many ways, as is the conceptualization of calculus-based gradient-descent. It must be remembered that while these paradigms are incredibly successful within their respective domains, both these paradigms are just that – *paradigms* for conceptualizing solutions in a computational search space. The interpretation back to the original source of inspiration becomes secondary to the results produced in the field of computation problem-solving and the results of this paper show great potential in this regard.

7. Future Work

There are many future possibilities for a gradient descent technique. As regards the specific GDM operator created in this paper, several modifications could provide a more robust and better-performing GA. First, the tuning parameters for this operator should be explored. Follow-up mutation could occur randomly for example, or on the *right* side of the initial position instead of the *left* side as implemented in this paper. The maximum number of follow-up mutations could also be varied to produce different results.

Its important to note that for any given individual, the likelihood is that only 1 position along its length would be mutated by GDM. Many calculus-based methods perform gradient-descent in all available dimensions of the search space, which would correspond to all genes in the individual. The possibilities of such an *n-dimensional* GDM (as opposed to the *1-dimensional* GDM that is executed in this paper) is described below.

Figure 7 below illustrates the potential of a GDM operator that mutated in 2 genes within the same individual. In this example, each of the 2 genes (4-bit length) starts with initial value of `0001` (for simplicity). The search space is shown as vector values from 0 to 15 (corresponding to binary values of a 4-bit string) and positions in this search space that individuals produced by GDM are shown. Note that the position of the original string is marked by the symbol "**O**", while the possible follow-up mutations are marked by the symbol "**X**". The right side shows a 2-dimensional implementation of GDM (both genes mutate through GDM) and this covers far more possibilities than the original version implemented in this paper. Additionally, it can be seen that if an individual was created through other genetic operators that is in the vicinity of a local minimum, a modified GDM operator can hone in on possible optimal locations.

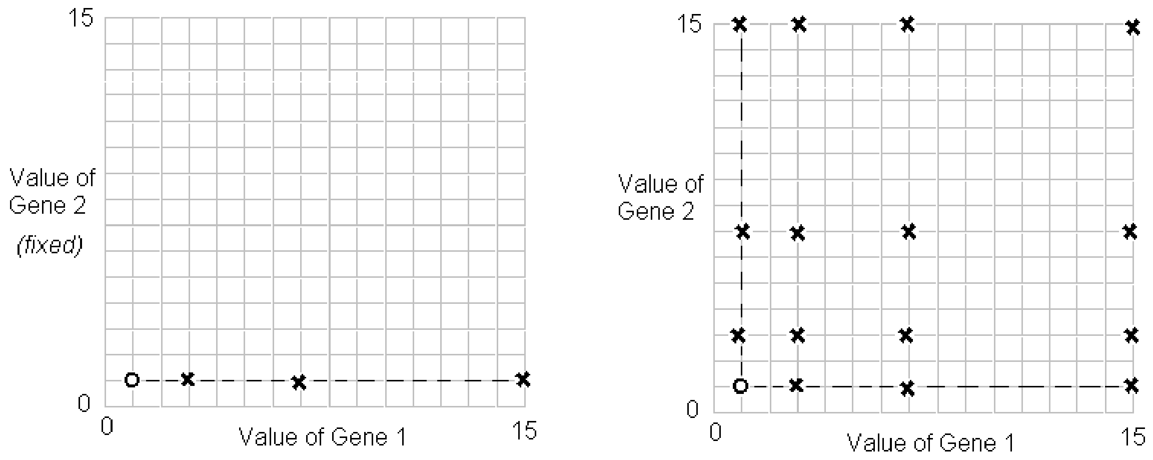


Figure 7 – Mutations from 1-dimensional GDM (left) and 2-dimensional GDM (right)

The concept of a gradient-descent genetic operator can be applied to many areas including GP. Possibilities include but are not limited to:

- Developmental GP where values of a parameter are incremented (or decremented) on a gradient-descent basis
- Incrementing a random ERC value using gradient-descent
- Performing follow-up mutation by replicating a node in a sub-tree similar to the manner in which GDM replicates a bit value along the chromosome length.

A comparison between GDM to simulated annealing would be an interesting future extension to this work. The distinction between the two must be made qualitatively – GDM is a *genetic* gradient-descent technique (that benefits from and contributes to other genetic operators such as crossover) while simulated annealing is a *probabilistic* gradient-descent technique. Data-driven comparisons such as those suggested here are necessary to identify the full potential of such a hybrid technique.

8. Bibliography

De Jong, K. A. 1975. *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral Dissertation. University of Michigan.

Grefenstette, John J. 1990. *Genesis*. Genetic algorithm software in C.

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley Longman Inc.

Koza, John R. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: The MIT Press.