Genetic Algorithms in Optimization: Better than Random Search?*

José Nelson Amaral, Ph.D.
amaral@ee.pucrs.br
http://www.ee.pucrs.br/∼amaral

Adalberto Teixeira Castelo Neto
castelo@ee.pucrs.br
http://www.ee.pucrs.br/∼castelo

Alessandro Valério Dias
dias@ee.pucrs.br

Programa de Pós-Graduação em Engenharia Elétrica
Pontifícia Universidade Católica do Rio Grande do Sul
90619-900 - Porto Alegre - RS - Brasil

Abstract

In this paper we show that bit strings are seldom the representation of choice for individuals in Genetic Algorithms and that genetic operators must be tailored to each specific problem. We use simple functions to compare the performance of bit string representation with the performance of a simple "blind" random search which requires the same level of computational effort as the GA. For approximation of continuous functions mapping R^n into R^1, we compare the performance of bit string GAs with that using standard floating point representation for the individuals. Experimental results indicate that the bit-string GA fares no better than the random search algorithm and that a GA using a floating point representation yields better results for complex search spaces.

*This research is supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), by Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) and by Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS).

1 Introduction

Combinatorial Optimization Problems (COP) are commonly found in all branches of engineering [9, 10]. Although most COPs are NP-complete problems [5], approximate solutions are often found in engineering through the reduction of a COP to a search in a high dimensional space [8]. Genetic Algorithms are often considered for search problems in which the solution space is described by non-continuous or multimodal functions [2, 7, 11, 12].

Because Goldberg used bit strings to represent individuals in his seminal work [6], many applications of Genetic Algorithms for COP in Engineering are still constructed using bit strings to represent individuals. In these applications, two-point crossover is often used for reproduction and mutation is accomplished through bit flipping [13].

In this paper we present a comparative study between genetic algorithms operating with bit strings and numerical representation of data. Our results indicate that unless a appropriate genotypical representation is used, GAs fail even in the simplest search problems.

2 Motivation

While participating as invited speaker in a joint Mexico-USA conference in 1995 [1], Amaral's attention was caught by a student paper that reported a simple study of GAs [3]. The students were seeking to minimize $f(x)$ given below on the interval $[0, 50]$.

$$f(x) = 1 + \frac{\cos(x)}{1 + 0.01\,x^2} \tag{1}$$

The function $f(x)$ is a continuous and multimodal function with eight local minima in the interval of interest. Even the simpliest GA should easily solve this problem. Moreover, if a simple GA fails to solve this problem we should have little hope about using GA for complex COPs; however, the students reported that in their first attempt to solve the problem, a simple GA with an initial population of fifty individuals that evolved for fifty generations failed to find the global minimum. After successfully obtaining the global minimum in a second trial using a population with a hundred individuals that evolved for a hundred generations, they concluded that GAs are sensitive to parameter tuning.

We decided to examine their solution closely because we were troubled by the fact that a simple GA was failing to solve such an easy problem. We noticed that a bit string was used as a genotypical representation of an individual. They first used a string of ten bits and then a string of twenty bits. The phenotypical representation of a individual was a real value x in the domain of the function $f(x)$. They used a single point crossover and their mutation operator complemented a single bit in the bit string representation according to a specified mutation rate.

3 The Problem with a Bit String Representation

Consider $g(x)$ of equation 2 plotted in Figure 2. This is a much simpler function then $f(x)$; therefore an algorithm that finds the minimum of $f(x)$ must be also able to find the minimum of $g(x)$. Let's consider a individual represented by a string of bits whose genotype is a point x in the domain of g. The fitness of each individual is the corresponding value $g(x)$. In this case the GA will be looking for the individual with the smallest fitness.

$$g(x) = x^2 \tag{2}$$

In order to further simplify this discussion, let's consider an integer representation for the individuals. Assuming that a individual represents a point in the domain of g in a ten-bit two-complement notation. For example, the binary representation for the numbers -2 and +1 are given below, with the respective fitness for these individuals.

x_1	=	11|11111110
x_2	=	00|00000001
		↓
x_3	=	11 0000001
x_4	=	00 1111110

Figure 3: Positional Crossover for the Minimum Seeking Example

$$x_1 = -2 = 1111111110 \Rightarrow f(x_1) = 4$$
$$x_2 = +1 = 0000000001 \Rightarrow f(x_2) = 1$$

We use a positional crossover operator and select an arbitrary position in the bit string to slice the individuals. Assuming that the individuals x_1 and x_2 were selected for reproduction, and that the position selected for slicing is to the right of the second bit, the offspring generated are shown in Figure 3.

In the integer two-complement notation that we are using, the resulting bit strings x_3 and x_4 represent the integer numbers -255 and $+254$, respectively. Computing the fitness function of the new offspring we find $g(x_3) = 3969$ and $g(x_4) = 3844$. Observe that we started with two individuals that were close to the best solution and ended up with two offspring that are quite far from the solution.

What we are seeing in this example is a very classical example of deception, which occurs

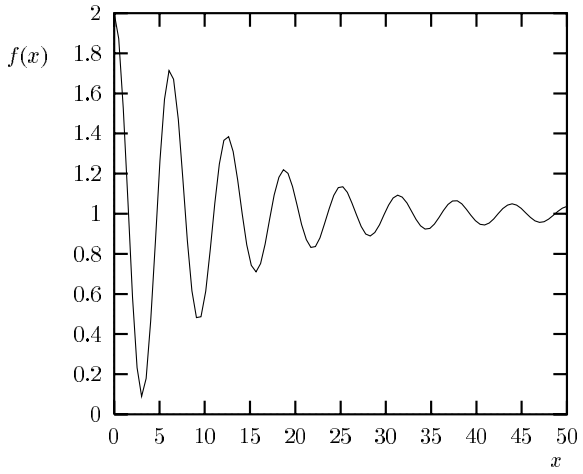

Figure 1: A graph of function $f(x)$ in interval $[0, 50]$.

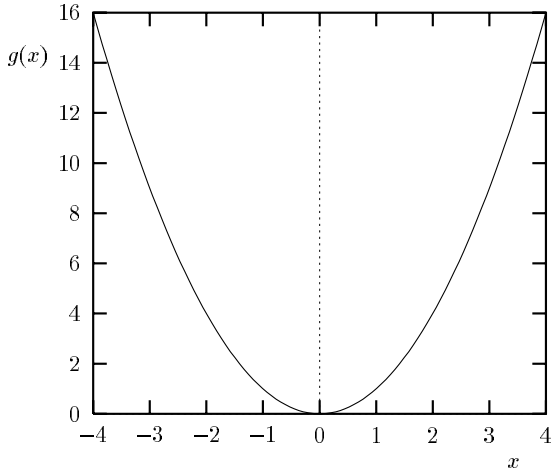

Figure 2: A graph of function $g(x)$ in interval $[-4, +4]$.

when the combination of two individuals that are close to the optimal solution results in offspring that are farther from it [4]. A better crossover mechanism for this problem simply computes the arithmetic average of the two parents, as illustrated in equation 3.

$$\begin{aligned} x_3 &= \frac{x_1 + x_2}{2} = \frac{-2+1}{2} = 0.5 \\ g(x_3) &= 0.25 \end{aligned} \tag{3}$$

For the parents selected earlier in this example, the offspring generated with this new crossover operator is better than either parent; however in some situations this new crossover operator is deceptive, as illustrated in Figure 4. The questions we have in front of us are how do we know whether a given crossover mechanism makes a certain problem deceptive to GA and how much deception can we accept and still be able to use GA to solve a problem. In most GA solutions, we must expect some degree of deception for most real life problem, i.e., we should expect that some applications of the crossover operators will generate offspring that are worse than either parent.

A second problem with the bit string representation of a individual is related to the mutation operator. In the Canonical Genetic Algorithm the mutation operator randomly selects one or more bits in the bit string representation and complements it [12]. Let's examine

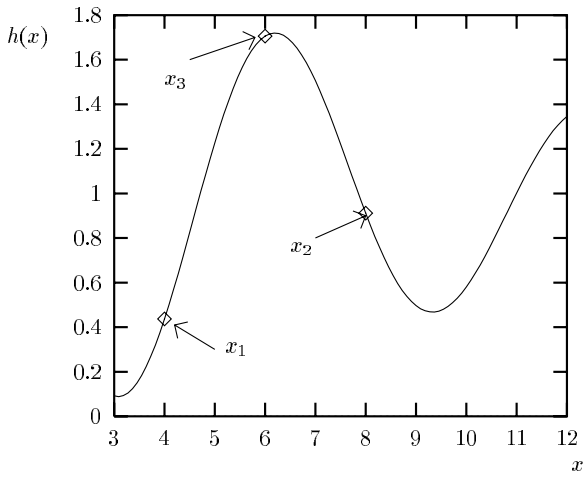

Figure 4: Deception with the new crossover operator.

the two applications of this mutation operator shown in Figure 5.

The problem ilustrated in Figure 5 is that two applications of the same mutation operator might produce very different amounts of change in the individual. A desirable property for a mutation operator is that the amount of change introduced by its application be controlled by a parameter called *mutation rate* [14]. One solution to this problem is to attach a probability of mutation proportional to the weight of each bit of the string. In this way the least significant bits would be more likely to be complemented than the more significant ones.

In section 4 we report experimental results that compare the quality of solutions obtained by two versions of GA and a "blind" random search. The objective is to investigate whether the problems with bit string representation of individuals that we discussed in this section actually affect the performance of GAs.

4 Experimental Results

After the analysis presented in section 3, we decided to implement a fair experiment to compare three approaches:

GA with bit string - Each individual is represented with a 32 bit string. An elitist selection strategy is implemented with a rank-based roulette wheel [12]. A two point string slicing crossover mechanism is used and a mutation rate of 30% indicates that whenever a new offspring is generated, there is a 0.3 probability that a random picked bit will be complemented. The results of this algorithm are labeled **string** in our graphs.

GA with numerical representation - The genotypical representation of a individual is a 32-bit floating point number in the interval between 0.0 and 50.0. Also uses an elitist selection through rank-based roulette wheel. The crossover mechanism is the one described by equation 3. Every individual undergoes mutation. The mutation rate specifies how much each individual changes in the mutation process. The results of this algorithm are labeled **numerical** in our graphs.

Random search - The random search method simply randomly generates 50 new individuals for each generation. The best individual is always kept in the next generation. Thus after y generations we have the best of $50 \times y$ randomly generated individuals. The results of this algorithm are labeled **random** in our graphs.

$$\begin{array}{lll} x_1 = 0000000001 & = +1 & \Rightarrow \quad g(x_1) = 1 \\ \downarrow & & \\ x_3 = 0000000011 & = +3 & \Rightarrow \quad g(x_3) = 9 \end{array}$$

$$\begin{array}{lll} x_1 = 0000000001 & = +1 & \Rightarrow \quad g(x_1) = 1 \\ \downarrow & & \\ x_3 = 1000000001 & = -511 & \Rightarrow \quad g(x_3) = 261121 \end{array}$$

Figure 5: Mutation Operations in the Minimum Seeking Example

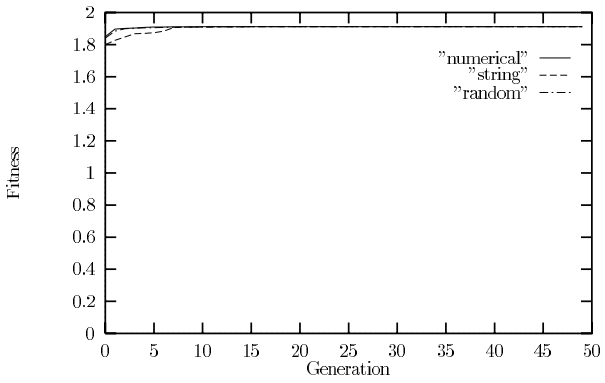

Figure 6: Performance of the three methods with function $f(x)$. The curves in this graph were obtained averaging the best individual of each generation over the twenty runs of each method.

4.1 Working with a Simple Solution Space

Observe that all three methods will evaluate the fitness of the same number of 50×50 individuals. Considering that most of the computational cost in the implementation of a genetic algorithm is incurred during the evaluation of the fitness, we might consider that the three methods have the same computational cost. For the GA that searches for the minimum of the function $f(x)$ we defined the fitness function according to equation 4. This definition allows the GA to search for the maximum of the fitness.

$$\text{fitness}(x) = 2 - f(x) \tag{4}$$

Our first experiments were with the function $f(x)$ described earlier. The results for the three methods are shown in Figure 6. Although the bit string version of GA performed slightly worse than the random search or the numerical representation, there is no relevant distinction between the results obtained for any of the methods. We should observe the the GA with numerical representation of individuals only performed to the same level as the random search and the bit string GA after some tuning of the mutation rate. It seems that if the mutation rate is too low the algorithm cannot always explore the entire solution space.

From this, we might conclude that regardless of our choice for the genotypical representation of an individual, the performance of GAs is the same of a simple random search that evaluates the same number of individual; however we must notice that this is a very simple function. Before jumping to a conclusion, we should study some problem with a more complex solution space.

4.2 What Happens When the Solution Space is More Complex?

To investigate how the two versions of GA and the random search compare in a problem with a more complex solution space we choose the function $h : R^n \rightarrow R$ presented by

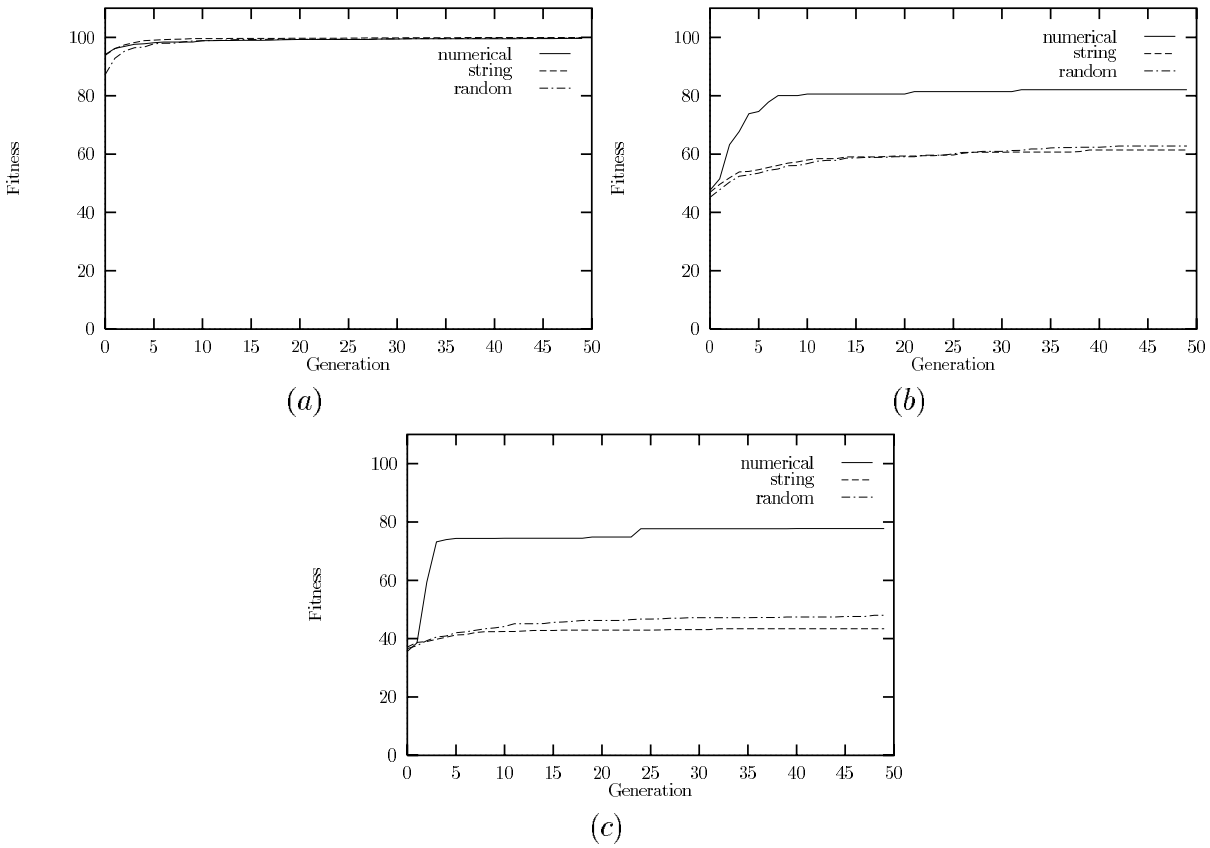

Figure 7: Comparative results of the three algorithms for the function $h(x_1, x_2, \ldots, x_n)$. (a) $n = 1$; (b) $n = 5$; (c) $n = 10$.

Tanomaru [13] and defined by equation 5.

$$h(x_1, x_2, \ldots, x_n) = \frac{100}{n} \sum_{i=1}^{n} \left(\frac{\cos(0.15\pi(x_i^* - x_i)) + 1}{2 + 0.0025(x_i^* - x_i)^2} \right) \quad (5)$$

The advantage of this function is that the complexity of the search space changes with the value of n. The graphs in Figure 7 present the results with n equal 1, 5, and 10 for the three algorithms. The careful reader will observe that our results for $n = 10$ are significantly worse than the ones published by Tanomaru [13]. Because we were interested in comparing the performance of the three methods with roughly the same design effort, we did not run as many generations as he did, and we did not spend much time tuning our numerical GA.

Observe that for $n = 1$, still a quite simple problem, all three algorithm have roughly the same performance, but as the search space becomes more complex the GA with a numerical representation for individual solutions outperforms the other two methods. Even with significantly more complicated problems, the GA that uses bit strings to represent individuals yields the same performance of a simple random search, and in some experiments even a slightly worse performance.

5 Conclusion

The results of the experiments presented in this article allow some insights in the performance of GAs. First, GA operators must be designed to fit the problem that we are trying to solve. An often repeated mistake is to replicate the operators originally published by Goldberg [6] producing a gross missfit with the problem we are trying to solve. Second, a poorly designed GA will perform no better

than a simple random search. Third, unless we use test cases with a search space that is complex, enough we might not observe the poor performance of our GA.

Finally, answering the question posed in the title of this article: Yes, well designed GAs can do better than a random search in the solution space.

References

[1] J. N. Amaral. Genetic algorithm and evolutionary neural computation. Invited Lecture in *Sian Ka'an International Conference: The First Joint Mexico-US International Workshop on Neural Networks and Neurocontrol Proceedings*, to appear, September 1995.

[2] J. N. Amaral, K. Tumer, and J. Ghosh. Applying genetic algorithm approaches to the state assignment problem. *IEEE Transactions on Systems, Man and Cybernetics*, 25(4):687–694, April 1995.

[3] A. L. O. Cruz, M. A. B. Saucedo, and J. L. P. Silva. Simple study of genetic algorithm. In *Sian Ka'an International Conference: The First Joint Mexico-US International Workshop on Neural Networks and Neurocontrol*, pages 212–226, September 1995.

[4] R. Das and D. Whitley. The only challenging problems are deceptive: Global search by solving order-1 hyperplanes, 1991.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to NP-completeness*. W. H. Freeman, San Francisco, 1979.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.

[7] D. E. Goldberg. Genetic and evolutionary algorithms come of age. *Communications of ACM*, 37(3):113–119, March 1994.

[8] H. Mühlenbein, M. Georges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421, 1989.

[9] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[10] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[11] W. M. Spears and K. A. De Jong. Using neural networks and genetic algorithms as heuristics for np-complete problems. In *International Joint Conference on Neural Networks*, pages 118–121, 1990.

[12] M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, June 1994.

[13] J. Tanomaru. Motivação, fundamentos e aplicações de algoritmos genéticos. In *II Congresso Brasileiro de Redes Neurais*, pages 373–403, 1995.

[14] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, 1989.