Visualization of a Simple Genetic Algorithm for Pedagogical Purposes

Vedrana Vidulin[1], Bogdan Filipič[2]

[1,2] *Jožef Stefan Institute, Department of Intelligent Systems, Jamova 39, SI-1000 Ljubljana, Slovenia*
E-mail: vedrana.vidulin@ijs.si, bogdan.filipic@ijs.si

Abstract. Genetic algorithm is an evolutionary search technique that is becoming increasingly popular in solving practical problems like timetabling, scheduling, engineering design, and other optimization problems. In this paper we present a computer program implemented to perform basic experimentation with a simple genetic algorithm with intention to gain understanding of how genetic algorithms work. The program allows changing the algorithm parameters, and shows their effects in a graphical form including solution encoding and graphs of the best-so-far fitness.

Key words: evolutionary computation, optimization, simple genetic algorithm, visualization, best-so-far fitness

1 Introduction

"A genetic algorithm (GA) is a search technique used in computer science to find approximate solutions to optimization and search problems. They are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination)." [4].

GAs are used in practice for different purposes, e.g. for solving timetabling, scheduling, engineering design and other optimization problems. Implementation, though, depends on the particular problem that needs to be solved. In this work we do not go into the details of a specific implementation. Instead, we describe a visualization computer program built on the basic form of a genetic algorithm, known as simple genetic algorithm (SGA).

The idea of visualizing the algorithm is taken from [8], while the implementation follows the description of the SGA coded in Pascal [5]. In a later work the algorithm was improved and implemented in the C programming language [10]. Both sources were very useful in our work, although certain adjustments were needed to implement the SGA in C# [2, 7, 9].

To facilitate the functioning of the SGA, a simple problem is used, i.e. a problem of a box with switches. There are ten switches on the box, and depending on which switches are on and off, the box produces a result. The target is to gain an optimal result by experimenting with turning the switches on and off, and in this case the optimal result is obtained when all switches are turned on.

To solve this problem using the SGA, we need to define the encoding of candidate solutions or their genetic representation, and a fitness function that is a measure of the solution quality. It is simple to encode the switching problem because the state of every switch can be represented with only two values, i.e. 0 when the switch is off, and 1 when the switch is on. A complete solution that includes the states of all ten switches can be encoded as a string of ten binary digits.

To calculate the fitness of a solution, we first need to decode the solution. Decoding is performed by summing powers of two as implied by the binary code. For example the decoded value of the solution 0010001011 would be $2^7 + 2^3 + 2^1 + 2^0 = 128 + 8 + 2 + 1 = 139$. Accordingly, the weakest solution would have the value 0, and the best or optimal solution would be 1023.

In our implementation we did not use such a straightforward implementation of a fitness function. Instead, the function

$$f(x) = 100 * (x / coeff)^n \quad (1)$$

was used, where n could be set as a parameter of the program with a default value of 10, and the value of the coefficient (*coeff*) depends on the length of the solution. The coefficient is determined as follows:

$$coeff = 2^m - 1 \quad (2)$$

where m represents the length of the solutions. In the 10-bit switching box problem *coeff* is 1023 and this number is used for the normalization purposes. The normalized value is then multiplied by 100, as we want to get fitness values ranging from 0 to 100, with 0 as the lowest and 100 as the highest fitness.

Solutions are evaluated in generations, and in our SGA implementation we chose generations of ten solutions. Solutions in the initial generation are generated randomly [6] bit by bit by flipping an unbiased coin, and in other generations are obtained by applying genetic operators, i.e. selection, crossover and mutation to the solutions of the last available generation.

Selection is an operation used for choosing the fittest solutions for reproduction [1, 3, 5]. From numerous methods of performing the selection in a genetic algorithm we implemented the so-called roulette-wheel

selection which uses "a biased roulette wheel where each current string in the population has a roulette wheel slot sized in proportion to its fitness… Each time we require another offspring, a simple spin of the weighted roulette wheel yields the reproduction candidate." [5].

On the selected solutions the operations of crossover and mutation are performed. Crossover implies exchange of information between two solutions. The crossover point is chosen randomly [1, 3, 5]. If, for example, it is between the third and the fourth bit, then as a result we get two new solutions where the first three bits are copied from parent solutions and other bits are exchanged, i.e. the first child gets other bits from the second parent, and the second child gets other bits from the first parent. Mutation means that some randomly chosen bits are changed from 0 to 1 or from 1 to 0 [1, 3, 5]. Probabilities of crossover and mutation are set as parameters of the program and are realized by flipping a biased coin.

2 Visualization

The purpose of the developed program is to graphically present the operation of the SGA and therefore facilitate its understanding. Two forms of graphical representation were implemented, i.e. the representation of solutions in the form of colored strings composed of zeros and ones, and representation of the best-so-far fitness in a graph (see Figure 1).

The first form of visualization presents two populations, the old one on which the genetic operations will be performed, and the new one that is a result of applying the genetic operations. Each individual solution in the old population is presented by different color. The solutions in the new population are colored in such a way that from the colors one can see which genetic operations were applied to them. For example, as a result of the selection, two solutions were chosen, the first and the sixth solution. The first solution in the old population is colored in red and the sixth

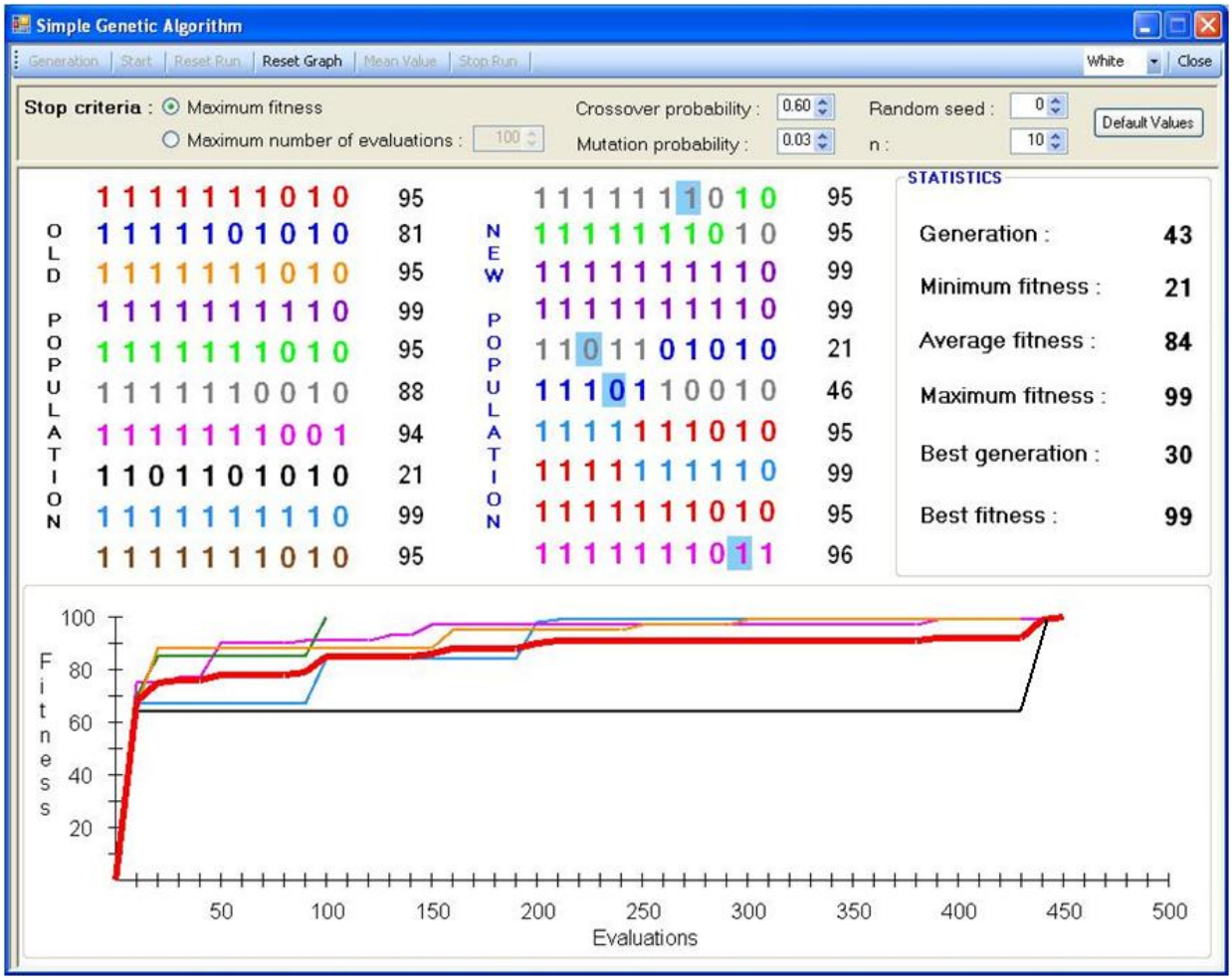


Figure 1. The SGA visualization program supports two forms of graphical representation, i.e. representation of solutions using colored binary strings, and best-so-far fitness values shown in a graph.

solution is colored in yellow. The crossover point is between the third and the fourth bit of the solution. The result of applying genetic operations will be represented in the new population as two strings, with the first three bits of the first string colored in red (showing that they are copied from the first parent) and other bits colored in yellow (copied from the second parent). Likewise, in the case of the second child, the first three bits will be colored in yellow and others in red. If somewhere in the process mutation has happened, then it is presented by coloring the background of the bit in blue.

The second approach to visualization is representing best-so-far fitness in the form of a graph. The best-so-far fitness is the highest fitness obtained in evaluating the solutions during the course of the algorithm run. On the x-axis, the number of solution evaluations is presented, and on the y-axis the fitness. Every run is presented by a line of different color, so that the results of different runs can be compared. In addition, the mean value of the best-so-far fitness over all runs can be calculated and shown in the graph as a thick red line (see Figure 1). Because of the clarity of the graphical representation, the number of runs traced in graphical representation is limited to ten. On the other hand, the feature of automatic graph shrinking makes it possible to represent long runs as well.

Visualization is accompanied by statistics showing the following characteristics of the algorithm run:

- Generation – Shows the number of generations in the current run.
- Minimum fitness – Shows the minimum fitness of the new population.
- Average fitness – Shows the average fitness of the new population.
- Maximum fitness – Shows the maximum fitness of the new population.
- Best generation – Shows in which generation the best fitness was achieved in the current run.
- Best fitness – Shows the best fitness achieved in the current run.

3 Program Functions

All program functions are placed on one screen. To work with the SGA we need to set some inputs or the parameters of the run. They are as follows:

- Stop criterion – We need to set some target when applying the SGA in problem solving. The target can be achieving the maximum fitness, i.e. optimal solution, or we can conduct a predefined number of evaluations (in the range from 1 to 10000) no matter of the achieved fitness value. As a default value of this parameter, the maximum fitness is chosen.
- Crossover probability – Defines the probability that two randomly selected solutions will undergo crossover. The default value is 0.6.
- Mutation probability – Defines the probability that a bit in the solution string will be mutated. The default value is 0.03.
- Random seed – A number used to calculate the starting value for the random number sequence. The Random seed can be a value between 0 and 100, and the default value is 0, which means that the starting value for the random number sequence will also be chosen at random.
- n – This integer parameter represents the exponent in the fitness function as defined by the Equation (1). Choosing higher values makes the search problem harder for the SGA (see Figure 2) and results in more solution evaluations needed for finding the solution. The value of n can be set between 1 and 1000. The default value is 10.
- Default Values – This button is used to set the default values of all program parameters.

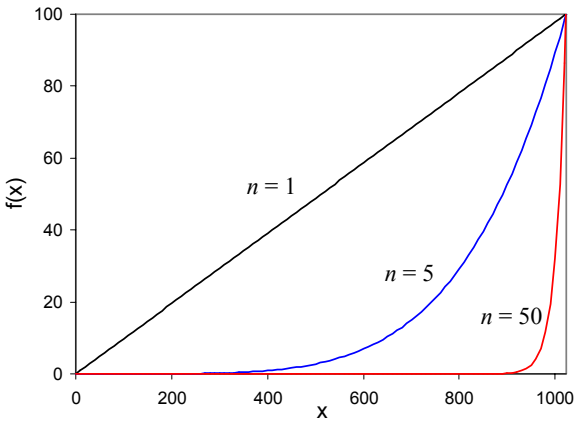


Figure 2. Effect of the exponent n on the fitness function. Higher values make it more difficult for the SGA to find the maximum.

After setting the parameters we can choose the action by clicking a button on the toolbar placed on the top of the screen. The buttons are as follows:

- Generation – By clicking on this button we evolve solutions generation by generation. This option allows for detailed monitoring of the results. It is based on the algorithm parameters set in the beginning of the run, what means that changing parameters before showing the next generation, except for the initial population, would have no effects. Similarly, changing the stop criterion parameter does not have any effect.
- Start – Starting the process of evolution. Solutions are evolved until the stop criterion is satisfied or the run is stopped by clicking on the Stop Run button.
- Reset Run – Used for preparing the program for a new run by resetting the populations and statistics. The primary function of this option is to reset the run conduced generation by generation.

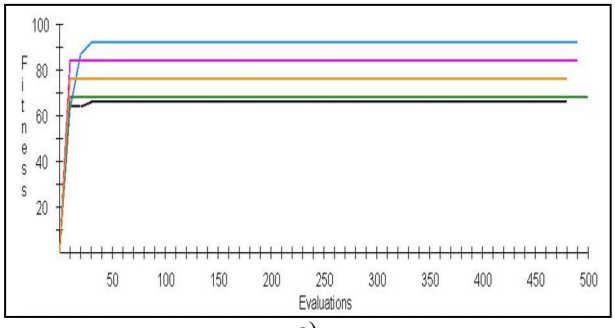
- Reset Graph – We use this option when we want to show a new set of runs on the graph. As a result of pressing this button, graphical representations of old runs in the graph are deleted.
- Mean Value – Shows the mean value of the best-so-far fitness for all runs presented in the graph. It is available after conducting two or more runs. The result of clicking on this button can be seen in Figure 1 as a thick line.
- Stop Run – Option available during the SGA run. By clicking on this button we stop the current run.
- Close – Used for closing the program.

On the toolbar, besides the buttons, there is also a drop-down box that enables us to choose between black and white color schemes.

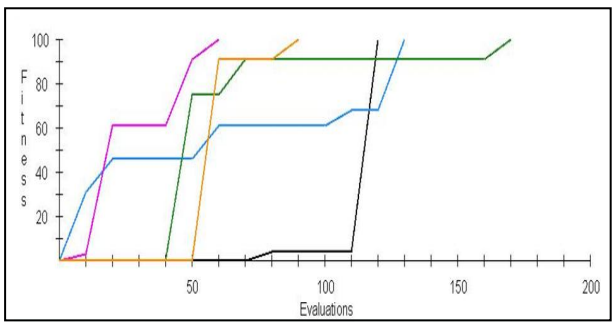
4 Example Situations

In addition to demonstrating the core functionalities of the SGA, such as genetic operations on candidate solutions, we can also show the effects of parameter changes on the algorithm performance. For example, we can set mutation probability to 0, meaning that there would be no mutations. As a result, after some initial progress the SGA will get stuck at a certain fitness value and unable to improve further (see Figure 3a).

Another illustrative situation is obtained by setting the fitness function exponent n to a high value which makes it harder for the SGA to find the optimum (Figure 3b).



a)



b)

Figure 3. SGA performance: a) at zero mutation probability, b) with the fitness function exponent $n = 1$

5 Conclusion

Visualization is a powerful tool that can be used for facilitating explanation of functionality of the SGA. In this paper we presented a computer program that implements two forms of the SGA visualization. One form uses colored strings to represent genetic operations applied to candidate solutions. The other form represents progress of the SGA by showing a graph of the best-so-far fitness. In this graph, multiple runs can be shown, and after conducting two or more runs, the mean value of their best-so-far fitness values can be calculated and drawn in the graph.

By setting different values of the algorithm parameters, e.g. probability of crossover and mutation, random seed etc., we change the behavior of the SGA, and the graphical representation of the algorithm performance is an ideal way of showing the influence of parameter changes.

The program was created for educational purposes and will be used in the beginner courses on evolutionary computation to demonstrate how the SGA works.

6 References

[1] M. Berthold, D. J. Hand, *Intelligent Data Analysis*, Chapter 10: Stochastic Search Methods, p. 351-401, Springer, 2003

[2] E. Brown, *Windows Forms Programming with C#*, Manning Publications, 2002

[3] A. E. Eiben, J. E. Smith, *Introduction to Evolutionary Computing*, Chapter 2: What is an Evolutionary Algorithm, p. 13-35, Springer, 2003

[4] Genetic Algorithm – Wikipedia, 2006, http://en.wikipedia.org/wiki/Genetic_algorithm

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Chapter 1: A Gentle Introduction to Genetic Algorithms, p. 1-25, Chapter 3: Computer Implementation of a Genetic Algorithm, p. 59-88, Addison-Wesley, 1989

[6] R. L. Haupt, S. E. Haupt, *Practical Genetic Algorithms*, 2nd Edition, Chapter 2: The Binary Genetic Algorithm, p. 25-48, Wiley-Interscience, 2004

[7] J. Liberty, *Programming C#*, O'Reilly Media, 2005

[8] M. Obitko, P. Slavik, Visualization of genetic algorithms in a learning environment, *Spring Conference on Computer Graphics SCCG'99,* Comenius University, Bratislava, p. 101-106, 1999

[9] S. Robinson, C. Nagel, K. Watson, J. Glynn, M. Skinner, B. Evjen, *Professional C#,* 3rd Edition, John Wiley & Sons, 2004

[10] R. E. Smith, D. E. Goldberg, J. A. Earickson, *SGA-C: A C-language Implementation of a Simple Genetic Algorithm*, The Clearinghouse for Genetic Algorithms, Technical Report No. 91002, University of Alabama, Department of Engineering Mechanics, Tuscaloosa 1994