Impact of the Number of Chromosomes on the Fitness Value Improvement in Standard GA Applications

Ivana Stojanovska[1], Agni Dika[2] and Blerta Prevalla[3]

***Abstract* – In this paper we make visual representation of the data we obtain from the applications that implement the GA, which clearly shows the fitness value minimization over the generations and the impact of the number of chromosomes on the fitness value and the time required to find the optimal solution.**

***Keywords* – Optimization, Chromosome, Fitness value, Generation, Visualization.**

I. INTRODUCTION

Genetic algorithms (GA) are search and optimization algorithms that use the theory of evolution as a tool to solve a problem in science and engineering. They incorporate the idea of survival of the strongest in a search algorithm that provides a searching method which does not necessarily needs to examine every possible solution in a given practice area to achieve good result. [18]

The main effect of GA is in the parallel nature of its search. They implement powerful form of "hill climbing" which supports multiple solutions, eliminate those who do not promise, and improve the best solution. [1] Fig. 1 shows several solutions that converge to the optimal points in the search space. Initially, the solutions are spread through the space of possible solutions. After several generations, they tend to group (cluster) around the areas with a higher quality solution.

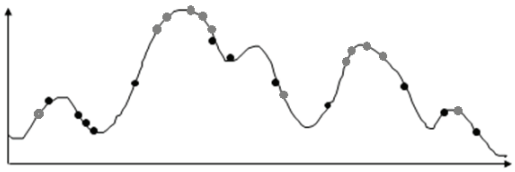


Fig. 1. Distribution of candidate solutions in Generation 1 (the black points on the curve) and Generation N (the gray points on the curve)

The process of GA generally consists of the following steps: Encoding, Evaluation, Crossover, Mutation and Decoding [18]. Once all of this is done, a new generation is evolved and the process repeats until they meet some stopping criterion. At this level the individual who is closest to the optimal solution is decoded and the process is completed.

[1]Ivana Stojanovska is with the Faculty of Information and Communication Technologies, Vojvodina bb, 1000 Skopje, Macedonia, E-mail: ivana.stojanovska@fon.edu.mk

[2]Agni Dika is with the Faculty of Contemporary Sciences and Technologies, Ilindenska bb, 1200 Tetovo, Macedonia, E-mail: a.dika@seeu.edu.mk

[3]Blerta Prevalla is with the Faculty of Information and Communication Technologies, Vojvodina bb, 1000 Skopje, Macedonia, E-mail: blerta.prevalla@fon.edu.mk

II. SOLVING THE TRAVELING SALESMAN PROBLEM

This problem is well known and is a standard problem for testing search algorithms of this type. The basic problem consists of the following: the traveling salesman is required to pass n given cities, but each city to visit only once. The applied algorithm has to find the minimum time for performing such a journey through the cities.

A. Problem Complexity

The traveling salesman problem is of particular importance because it is a classic example of NP-hard (non-deterministic polynomial time hard) problem, that so far can be solved only in exponential time. It is a classic problem with great computational complexity. If there are n cities, then the maximum number of possible plans to travel between towns is (n-1)!. You can create a simple algorithm which examines all possible paths and comes up with the shortest one. But the main problem is that the time required for algorithm execution grows with tremendous speed as the number of cities increases. If there are 25 cities, then the algorithm must look 24! possible routes. 24! is approximately 6.2• . Even if you use a computer that can investigate one million routes per second, it would take about 6.2• / = 6.2• seconds to solve the problem. This is approximately 1.96• years.

By using dynamic programming techniques the problem can be solved in $O(2^n)$ time. Although this solution increases exponentially, it is much better than O(n!).

B. Implementation

In our implementation we chose the encoding where each gene in the chromosome represents a city, and the chromosome represents the order in which the traveling salesman would move. We use an implementation with 25 cites and we must not forget that the salesman should visit each city only once.

8	2	3	4	7	6	1	5

Fig. 2. A chromosome which represents one of the possible solutions

The fitness function that characterizes each chromosome, represents the total length of the route from the first to the last

gene (city) moving according to the order of the genes in the chromosome. If the cities are represented with x and y coordinates in 2D coordinate system, then we calculate the distance between them according the Eq.1:

$$r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (1)$$

The fitness value of each chromosome is the sum of all distances between the genes and the goal of this GA is to minimize this function.

When starting the application it is necessary to initialize a starting population with a given number of chromosomes (in our application this number is 150). Once created, this population should provide a method for crossing into the next generation where these chromosomes are replaced with new chromosomes by applying the GA operators. The best solution from the current generation is saved and added to the new generation, if it does not already exist there. All this is repeated a number of times, which equals the number of generations in our application. The latest generation of chromosomes should provide the best solutions.

C. Graphical Representation

Data obtained from the program that implements the GA are visually presented in the form of a graph from which we can see the progress of the GA's fitness value minimization over the generations. Also shown is the data for the smallest fitness value, generation of its occurrence and the solution itself (the chromosome with the smallest fitness value). This application is developed in Gambas under Linux Ubuntu 7.04 and its appearance is shown in Fig.3.

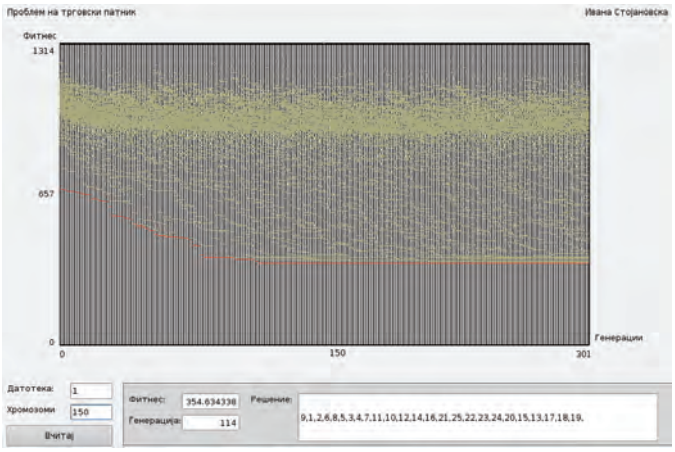


Fig. 3. Application for fitness value visualization over the generations

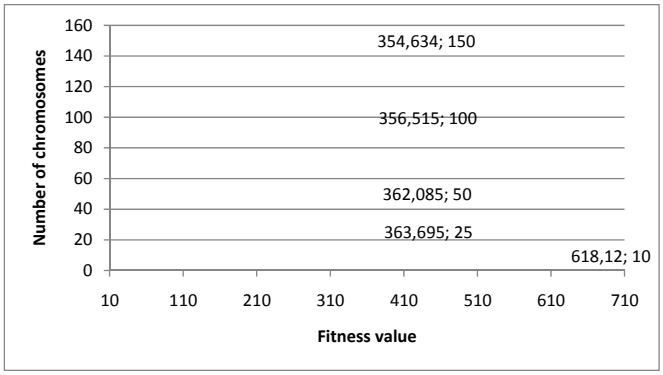
Initially, for the program execution we take population of 150 chromosomes, each composed of 25 genes (as the number of cities). For algorithm execution we use 300 generations to obtain the results shown in Fig. 3. From the figure we can see that the population is advancing toward a population with slightly better features. The gray cloud represents all the solutions in a generation, and in each of these generations the best solution is shown with a red line. As it can be seen in the figure, the best fitness value occurs in generation 114, with the best fitness value of 354.63 and the exact solution is: 9, 1, 2, 6, 8, 5, 3, 4, 7, 11, 10, 12, 14, 16, 21, 25, 22, 23, 24, 20, 15, 13, 17, 18, 19.

Within this visualization we made an analysis on the impact of the number of chromosomes on the fitness value and the time required to find the optimal solution. Moreover, we got the results shown in Table I.

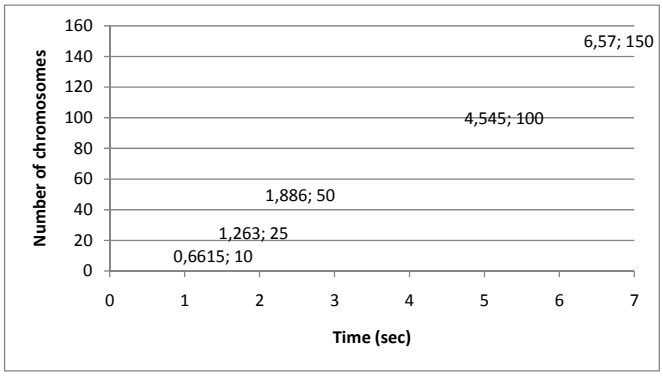
TABLE I
VARIABLE NUMBER OF CHROMOSOMES

No.of chromosomes	No.of generations	Fitness value	Time (sec)
10	300	618,120	0,661
25	300	363,695	1,263
50	300	362,085	1,886
100	300	356,515	4,545
150	300	354,634	6,57

Fig.4 (a), which illustrates the behavior of the fitness value when the number of chromosomes changes, shows that when the number of chromosomes is very small (10) the fitness value is significantly higher (618.120) according to the rest 4 examined cases (number of chromosomes = 25, 50, 100, 150) where it slightly improves. The graph in (b) illustrates that by increasing the number of chromosomes we get almost linear increase in time required for finding the optimal solution (the smallest distance between cities). The time increases slightly from 0.661, 1.263, 1.886, 4.545 to 6.57 sec, as the number of chromosomes gets bigger and bigger.



(a)



(b)

Fig. 4. Diagram of the influence of the number of chromosomes on the fitness value and the time

From the above analysis we can establish that when the number of chromosomes per generation is very small, it significantly affects the fitness value of the best solution found, but in further cases, there is almost no change in the fitness value of the best solution.

III. SOLVING THE 2D PACKING PROBLEM

The packing problem is actually a way of finding the optimal solution how to collect a given number of boxes (packages) in a large enclosed space. The problem is very easy to set and define, but it is quite difficult to solve. In mathematical terms speaking, as the traveling salesman problem, this problem is NP-hard, which means that when given a set of boxes and a space, it is very difficult to answer questions such as: Which is the best way to pack up the boxes, which would be the optimal solution and how good it would be. [19]

A. Problem Complexity

The packing problem is quite complex. When the number of packages is small the problem is relatively simple, but the complexity of the problem grows exponentially with the increase in the number of packages. For example, the number of possible combinations of 20 packages that can be oriented in 2 directions is: $(20 \cdot 2) \cdot (19 \cdot 2)...(1 \cdot 2) = 20! 2^{20} = 2{,}55108 \times 10^{24}$

First we have a choice of 20 packages that can be placed in 2 directions in the space, then we have 19 other packages to choose from, which can also be placed in 2 directions, etc.

If you perform a "brute force" test on this small problem of packing with 20 packages in 2D space, it will require a computer capable of checking millions of combinations of packings in a second, and still will need more than 10 years to complete the process. However, because GA use evolution to improve the solution, it relatively quickly starts to improve and becomes significantly better for a short time (several minutes), even when performing on relatively weak computer.

B. Implementation

In our implementation we selected the encoding where each gene in the chromosome is a package (number of packages equals 20), together with its orientation, and the chromosome represents the order in which they should be packed in the enclosed space (as shown in Fig. 5). The second field of each gene in the chromosome indicates whether the corresponding package should be rotated or not. If this field is set to true then the corresponding package is rotated from its initial position, and otherwise (false) it remains in the same orientation.

0	true	9	false	8	true	1	false	3	false	2	true	4	true	7	false	6	false	5	true

Fig.5. A chromosome which represents one of the possible solutions

The cost of each chromosome is calculated as the space wasted in the enclosed space, after all the packages are packed according to Eq.2:

$$\cos t = 1 - \frac{a}{b \cdot c} \tag{2}$$

where b is the highest point on the top package in the enclosed space, c is the width of the space, and a is the total area of the packages. So the optimal solution would have a cost of 0, if the area of the space is equal to the area of the boxes (packages).

At the start of the application we initialize a starting population with randomly generated chromosomes. As input, the application receives a .txt file that contains information about the height and the width of the packages. [19] Once created, this population provides a method for crossing into the next generation where these chromosomes are replaced with new chromosomes by applying the GA operators. All this is repeated a number of times and the latest generation of chromosomes should provide the best solutions.

C. Graphical Representation

As with the program for travelling salesman, data obtained from the program is visually represented in graph from which you can see the progress of the GA. Initially, for the execution of the program we take a population of 100 chromosomes, each composed of 20 genes (as the number of packages). During the execution it can be seen that the population gradually advances towards a population with better features. For algorithm execution we use 50000 generations to obtain the results shown in Fig.6. During the execution it can be seen that the population gradually advances towards a population with better features. It may be noted that after 50000 generations we obtain a fitness value of $\approx$8.9%, which is found in the 9069th generation and the exact solution is: 16, 0, 4, 12, 14, 15, 9, 1, 8, 17, 3, 13, 6, 7, 11, 5, 2, 10.

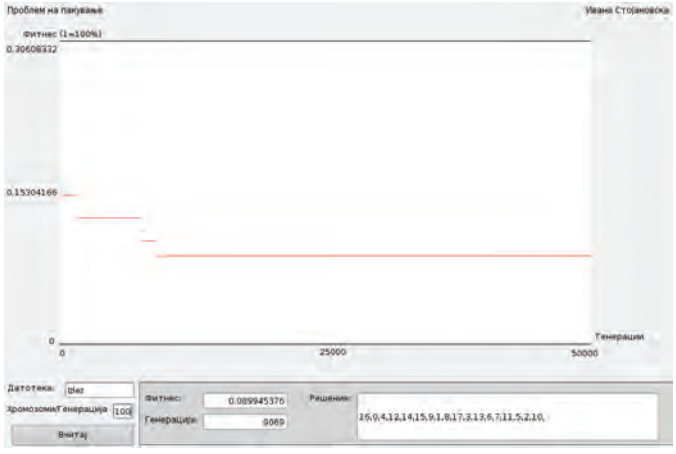


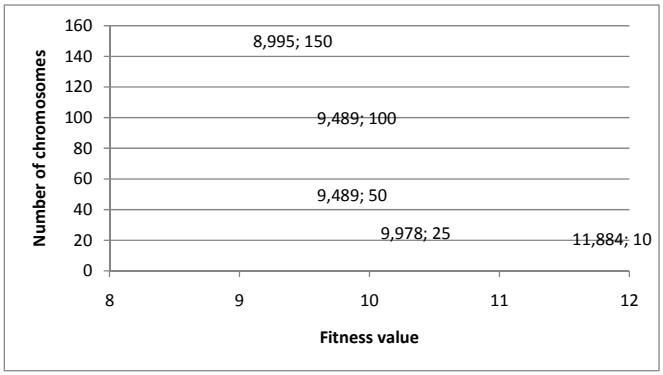
Fig. 6. Application for fitness value visualization over the generations

Fig.7 is made based on analysis conducted on the impact of the number of chromosomes to the fitness value and the time required to find the optimal solution. Within this analysis were obtained the results shown in Table II.

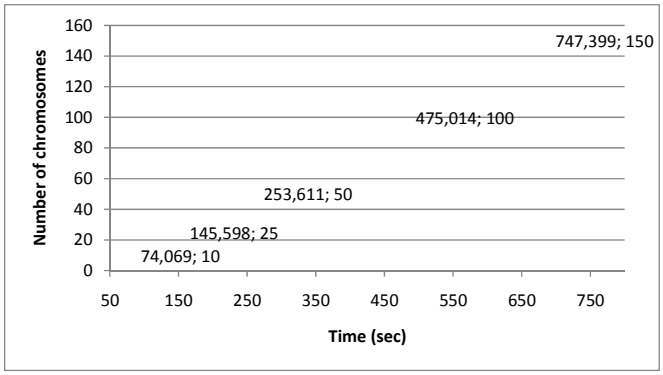
TABLE II
VARIABLE NUMBER OF CHROMOSOMES

No. of chromosomes	No. of generations	Fitness value	Time (sec)
10	50000	0,11884	74.069
25	50000	0,09978	145.598
50	50000	0,09489	253.611
100	50000	0,09489	475.014
150	50000	0,08995	747.399

Fig.7 illustrates the dependence of the fitness value and the time of the change in the number of chromosomes. At the graph in (a) it can be seen that when the number of chromosomes is very small (10) the fitness value is significantly higher (0.11884) in comparison to the rest 4 cases examined (25, 50, 100, 150) where it slightly improves as the number of chromosomes increases. The graphs in (b) shows that we have almost linear increase in time required for finding the optimal solution (packing with the smallest waist space). The time gradually increases from 74.069, 145.598, 253.611, 475.014,to 747.399 sec.



(a)



(b)

Fig. 7. Diagram of the influence of the number of chromosomes on the fitness value and the time

It may be concluded that increasing the number of chromosomes significantly affects the fitness value of the best solution found, meaning that as many generations we have as more likely that we will find a solution with minimal waist space on packing.

IV. CONCLUSION

With regard to what GA do best, which is an iterative improvement of the result, the conclusion has to be that the problems based on search, as the problem of packing and the traveling salesman problem, there are few other tools that are able to compete with the GA. One very interesting question in the analysis of this class of problems is whether it is worth spending many hours on expensive workstation to obtain a solution close to the optimum, or to work a few minutes on cheap personal computer (PC) to get "good enough" results for these applications.

REFERENCES

[1] George, F.L. (2005). Artificial Intelligence: Structures and Strategies for ComplexProblem Solving. Harlow, England: Addison Wesley.

[2] Mitchell, M. (1996). An introduction to Genetic Algorithms. London, England: MIT Press.

[3] Leardi, R. (2003). Nature-inspired methods in chemometrics: genetic algorithmsand artificial neural networks, Volume 23 (Data Handling in Science and Technology). Elsevier Science

[4] Mennon, A. (2004). Frontiers of Evolutionary Computation (Genetic Algorithms and Evolutionary Computation). Springer-Verlag

[5] Coppin. B. (2004). Artificial Intelligence Illuminated. Jones & Bartlett Publishers

[6] Steele, N.C. (2005). Adaptive and Natural Computing Algorithms: Proceedings of the International Conference in Coimbra, Portugal, 2005. Springer-Verlag

[7] Poli, Riccardo and Langdon, William B. and McPhee, Freitag N. (2006). A Field Guide to Genetic Programming. Lulu.com - Page 141

[8] Hauot, R.L., and Haupt, S.E. (2004). Practical Genetic Algorithms. Wiley-Interscience

[9] Zheng, Y., and Kiyooka, S. (1999). Genetic Algorithm Applications.

[10] Whitley, D. A genetic Algorithm Tutorial. Computer Science Department, Colorado State University.

[11] Holland, J.H. (1975). Adaptation in Natural and Artificial Systems. Press, University of Michigan.

[12] Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley.

[13] Pelikan, M., and Lobo, G.F. Parameter-less Genetic Algorithm: A worst-case Time and Space Complexity Analysis.

[14] Grefenstette, J.J. (1989). Genetic algorithms for changing environments.

[15] Grefenstette, J.J. (1989). How Genetic Algorithms work: A critical look at implicit parallelism.

[16] Herrmann, W.J. A Genetic Algorithm for Minimax Optimization Problems. Department of Mechanical Engineering and Institute for Systems Research, University of Maryland

[17] Sengoku, H., and Yoshihara, I. (1993). A Fast TSP Solution using Genetic Algorithm. Information Processing Society of Japan 46th Nat'l Conv.

[18] Bryant, K. (2000). Genetic Algorithm and the Traveling Salesman Problem.

[19] Sorensen, J.J., and Mortensen, H. The use of Genetic Algorithms on The Bin PackingProblem.

[20] Coley, D.A. (1999). An Introduction to Genetic Algorithms for Scientists and Engineers. World Scientific Publishing.