Date of publication DEC-31, 2021, date of current version OCT-19, 2021.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.20.4.2442

Aspects of the Study of Genetic Algorithms and Mechanisms for their Optimization for the Travelling Salesman Problem

NATALIYA BOYKO, ANDRIY PYTEL

Lviv Polytechnic National University, Lviv 79013, Ukraine (e-mail: Nataliya.I.Boyko@lpnu.ua, pytelandriy@gmail.com)

Corresponding author: Nataliya Boyko (e-mail: Nataliya.I.Boyko@lpnu.ua).

ABSTRACT Lately, artificial intelligence has become increasingly popular. Still, at the same time, a stereotype has been formed that AI is based solely on neural networks, even though a neural network is only one of the numerous directions of artificial intelligence. This paper aims to bring attention to other directions of AI, such as genetic algorithms. In this paper, we study the process of solving the travelling salesman problem (TSP) via genetic algorithms (GA) and consider the issues of this method. The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that are based on natural selection, the process that drives biological evolution. One of the common problems in programming is the travelling salesman problem. Many methods can be used to solve it, but we are going consider genetic algorithms. This study aims at developing the most efficient application of genetic algorithms in the travelling salesman problem.

KEYWORDS genetic algorithms; cross-breeding; crossover; mutation; generations; individuals; selection; evolution; travelling salesman problem; city; route.

I. INTRODUCTION

ONE of the common problems in programming is the travelling salesman problem. Many methods can be used to solve it, but we are going consider genetic algorithms. This study aims to design the most efficient application of genetic algorithms in the travelling salesman problem [1, 2].

Lately, artificial intelligence has been becoming increasingly popular. Still, at the same time, a stereotype has been formed that AI is based solely on neural networks, even though a neural network is only one of the numerous directions of artificial intelligence. This paper aims to bring attention to other directions of AI, such as genetic algorithms [3, 4].

We study the process of solving the travelling salesman problem (TSP) via genetic algorithms (GA) and consider this method's issues.

II. LITERATURE ANALYSIS

The origin of the travelling salesman problem is unclear. A handbook for travelling salesmen, published in 1832, mentions the problem and includes examples of tours in Germany and Switzerland, but contains no mathematical treatment [5-7].

It was first considered mathematically in the 1930s by Merrill M. Flood, trying to solve a school bus routing problem. Hassler Whitney at Princeton University introduced the name travelling salesman problem soon after [8, 9].

There are many methods of solving this problem. Some of them give exact results, others only approximate. One of the most interesting methods is the method of route optimization with the use of genetic algorithms [10, 11].

Nils Aall Barricelli conducted the first experiments, which involved simulated evolution, in 1945. Later the

experiments were also conducted by Nils Aall Barricelli, Ingo Rechenberg and Hans-Paul Schwefel. Thanks to artificial evolution a well-known optimization method appeared [1].

The article "Development, a new mutation operator, to solve the travelling salesman problem by the aid of genetic algorithms" of Murat Albayrak, Novruz Allahverdi [12] deals with the study of the method of Greedy Sub Tour Mutation for solving the Traveling Salesman Problem. The developed GSTM operator was tested with simple GA mutation operators. Two different greedy search methods and a component that provides distortion in this new operator are directly considered. The application of this GSTM operator gives much more effective results regarding the best and average error values.

In "Genetic algorithms for the travelling salesman problem", Jean-Yves Potwin [13] considers a classic combinatorial optimization problem that is easy to formulate but very difficult to solve. The problem is to find the shortest tour through a set of N vertices so that each vertex is visited exactly once. It is known that this problem is NP-complex and cannot be solved exactly in polynomial time. In this paper, the author uses a particular problem structure to develop complex crossover and mutation operators to encode the matrix chromosome. These operators are designed to support the ability to implement the solution during the search.

III. MATERIAL AND METHODS

A. TRAVELLING SALESMAN PROBLEM

The travelling salesman problem is a problem of finding the fastest (most efficient) route between n cities where the way must go once through every city. If the problem is presented in the form of a graph, then the answer will be the shortest Hamiltonian cycle.

The travelling salesman problem can be presented as in [4, 14]:

- Graph. In this form, the represented cities are displayed as vertices, and edges represent the criteria of profitability (distance, time).
- Asymmetric and symmetric problems. The catch in the asymmetric problem is that the profitability between the cities is dependent on the direction of edges, whereas, in the symmetric problem, the law has no role.

There are two main groups of methods for solving the travelling salesman problem, which can be combined [3, 15, 16]:

- Precise — they find the precise optimal solution to the problem but take a long time to calculate.
- Heuristic — they give an approximation of the optimal route but take notably less time to calculate.

Genetic algorithms are generally more efficient than the complete vocabulary as there is no need to go through all the possible combinations. At the same time, genetic algorithms are heuristic that does not guarantee a precise solution, but only the best possible approximation with the given amount of time (iterations) [17, 18].

B. GENETIC ALGORITHMS

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that are based on natural selection, the process that drives biological evolution.

Genetic algorithms can be broken down into the following steps [19, 20]:

1. Creation of the base population.
2. Iterations of the same actions (Evolution)
   - Rating.
   - Selection.
   - Cross-breeding and/or mutation.
   - Formation of the new generation in the case the result was not achieved.
3. Obtaining the resulting generation presented in Fig. 1.

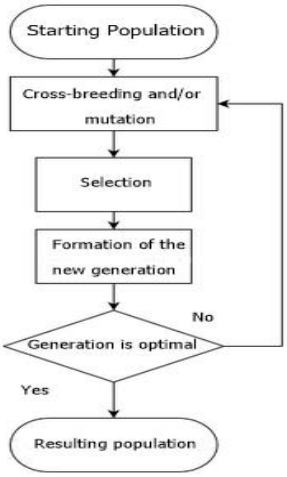

Figure 1. Stages of the genetic algorithm

During the creation of the base, population individuals are generated with mostly randomized parameters. Each parameter is a specific gene, and a set for those parameters form a chromosome. In our case, the most efficient type of data to store in a chromosome are route options. For instance, we have 5 cities A, B, C, D and E; inside the chromosome-specific routes, e.g., ABCDE, BADCE, BDCEA.

With the base population ready, we can start a cyclic process for the goal of creating a population that would become more and more optimized with each iteration.

The first step is for the population to go through a rating process where a specific adaptability score is assigned for each individual.

Now we can commence our selection. We do this by selecting the best-fitted individuals for their cross-breeding and/or mutation [6, 21].

To improve the level of adaptivity of the population in classic genetic algorithms, the best individuals are bred or mutated and sometimes both.

The process of cross-breeding imitates sexual reproduction. To create a new individual (child), 2 individuals must be called parents of the child. To inherit parent traits, child chromosomes are formed in a certain way by combining parent genes. This process is called a crossover.

When both mutation and cross-breeding have been performed to avoid a situation in which the population did not improve or got stuck on a particular iteration, some, if not all, individuals are mutated. In other words, one or some genes in a chromosome are replaced. Even though this mutation process is not mandatory as mutations can improve the rate of approximation to the result and slow it down [5, 20].

After cross-breeding and mutating, it is mandatory to correct the number of individuals in the population not to increase in size after each iteration. Only the most adapted individuals will go on into the next generation (the rest will be annihilated).

If the generation is not adapted enough, we repeat rating, selecting, cross-breeding/mutation and forming the new generation.

If the new generation is adapted enough, it can be considered to be adjusted. In our case, the route is as short as possible, and any improvements are insignificant or impossible [1, 10, 21, 22].

C. ANALYTICAL PART

The complexity of using this method is that it is not possible to use a regular crossover or a mutation.

Crossovers are generally presented as simple combinations of different parts of the parent chromosomes in classic genetic algorithms. For instance, let us say we have the following parents:

Parent 1 Parent 2

Then our children can have the following appearance.

Child 1 Child 2

As we can see, the regular crossover does not apply to this problem as it is possible to get to one city two times and do not get to another at all.

The same situation will occur with the use of the expected mutation.

This situation can be avoided if the complexity of the crossover and the mutation is increased.

D. THE FIRST STRATEGY (SIMPLE)

As mentioned before, combining halves of parent chromosomes will not be sufficient to avoid getting to the same city twice merely differently. The simplest way to resolve this is to change the city, which is repeated to a city that is not present. But at the same time, this method entails that each possible descendant may have several options.

Back to the same example.

Parent 1 Parent 2

Is incorrect. After the correction, they will have the following representation:

Child 1 Child 2

The main drawback of this method is the large number of iterations required to find the repeats that are needed to be completed for this cross-breeding. As a result, this method is relatively inefficient in terms of runtime.

Even though correcting children is similar to mutation, a full-fledged mutation still needs to be implemented because otherwise, child elements with an identical genome might cease to evolve.

E. THE SECOND STRATEGY (CYCLE)

The previous strategy was based on a basic crossover that had to be improved so that children could form a Hamiltonian cycle that would create a large quantity of children variations.

If we admit the premade movement directions taken from parent chromosomes, then there would be no reason to correct the children. As a result, by using this method, we acquire a wanted amount of children and we do not depend on the number of repeating cities.

The idea of this strategy is that the combination of parent genes should immediately form a Hamiltonian cycle [3, 5, 14, 7].

In addition to ensuring that the child chromosome would not repeat its parent, it is necessary to limit the length of the parents' chromosome from which the child will be built.

To understand this method, let us look at the following example. We have our parents:

Parent 1 Parent 2

Suppose we are allowed to copy up to three genes in one parent in a row.

To begin with, let us take a part of genes from one of the parents, three genes from parent 1 to be exact.

Then we shall take the other fathers' gene (BCAED). Next, we choose a direction leading to a city in which we have not been yet. And because we are looking for a cyclic route, we are not restricted to only moving from the previous gene to the following gene(city), but we can also move from the first to the last one and vice versa. In this case, we can both move to D and C. For the sake of optimization, let us assume that moving to the right has a higher priority, so we add C.

After that, we will be moving through the parent in the same direction until we hit a city to which we have been before. Then we move to the next parent and repeat the previous step. This continues until we reach all the cities.

In our case, after C comes A, to which we have been before, we also cannot move backwards, and because of that, we move to the next parent. Here we can go to D, and that is

going to be our last gene. As a result, we get:

E	A	B	C	A

F. THE THIRD STRATEGY (NEAREST)

Based on the second strategy, a new, more efficient crossover can be created. In the second strategy, we have up to 4 different moves. If the most efficient move can be chosen, then the runtime of the algorithm can be substantially improved while at the same time drastically decreasing the number of generations needed. However, the complexity of choosing the most efficient moves will increase the time designated for computing in a single age.

To make sure that the complexity of the algorithm will not increase, the restriction regarding copying the second parent's sequence shall be ignored. As a side effect, this creates an issue where the child can become a carbon copy of the parent: yet, it can be solved by either mutating such child or by banishing it from the general population.

Going back to our previous example, our parents:

E	A	B	D	C

Parent 1

B	C	A	E	D

Parent 2

In this case, we are not just going to the first available city, but instead, we choose an optimal path. A starting gene is randomly chosen, so for simplicity sake, in this example, the first parent gene will be selected, let it be E. Next, we have the options between A, C or D, and the shortest one will be chosen. Given that we have not specified the distances between the cities, we assume that C is the fastest. And as a result, we get more children who are more efficient than their parents.

G. MUTATION

To simplify the work, we will take the mutation method from the first method, i.e., the process of mutation follows the process for a standard genetic algorithm. Then we will correct the cities which are missing. As a result, we swap the places of two random genes.

The chromosome before mutation:

B	D	A	E	C

and after mutation:

B	C	A	E	D

We could also look for the most efficient swap to improve the algorithm, but that would only waste resources as it will amount too little to optimization gain.

H. SELECTION OF INDIVIDUALS FOR REPRODUCTION

The next problem is the selection of individuals who will be bred and the way how they will be bred. The most straightforward and efficient method is sorting the individuals according to some coefficient and pairwise crossing of the most adapted individuals [9, 22].

The fact is that crossing the same parents will yield the same individual. To avoid breeding the same pairs of parents, each individual will have its list of partners with whom it has already created offspring and its ability to generate with these individuals will be blocked.

Development of the software solution

The structure of the software solution can be separated into three parts [23]:

- Structural specifics of the travelling salesman problem are described in the structural interpretation of cities and connections between them.
- The genetic algorithm is the central aspect of this problem because the algorithm is described inside it.
- Cross-breeding and mutation implementation.

Structural specifics of the travelling salesman problem.

To work with genetic algorithms, input data is required, and in this particular problem, it is presented in the form of a list of cities between which we are looking for the shortest route.

To consider the graphic formulation of the route, cities are represented as points with their names on the coordinate plane.

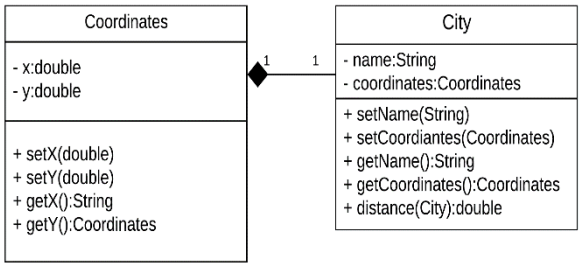

Figure 2. UML diagram of the city class

When selecting the structure to hold the data, it is essential to consider that connections between the cities are more important than the cities themselves (Fig. 2). In addition, it is wiser to store the calculated distances and retrieve them rather than constantly recomputing them. So, for this purpose, we will realize a sort of map which will keep all the cities and spaces between them. To ease the access to data inside the map, the list of cities and the distances between them will be presented as hash maps. Cities will be accessed through their names, and the spaces will be accessed by using a key that encodes the characters of cities and distances between them [15].

Because the algorithm is supposed to work with only one instance of the map, thus, we should restrict its access to creating multiple maps. This can be done by implementing a map based on the Singleton pattern.

Also, do not repeat creating a map every time we call the algorithm, options of saving the map to a file, loading it from the said file and generating it with randomized data have been implemented – the process of developing a map (Fig. 3).

Genetic algorithm in this specific problem

An individual is the smallest structural unit in any genetic algorithm, and all individuals have a chromosome and an adaptivity score. In our case, a chromosome is a sequence of cities, a.k.a is one of the possible routes and the adaptivity score is the ratio of the shortest known route to the length of

the current individual [5].

To determine the best-adapted individuals, we need to compare them by their adaptivity score, and for this, a "Comparable" will be used.

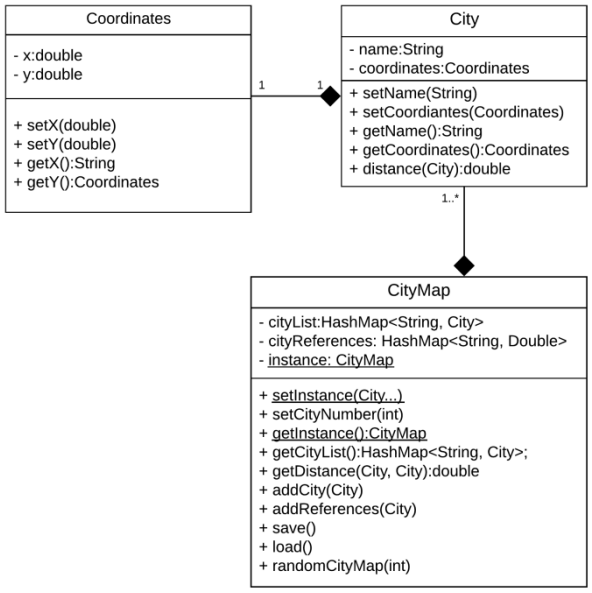

Figure 3. UML maps and their dependencies

The first generation will be generated with random routes.

We get our list of cities from the map, and for this, every individual should also have access to the map. Individuals should also have the same access to the values of the shortest route and the size of their chromosomes (Fig. 4) [6].

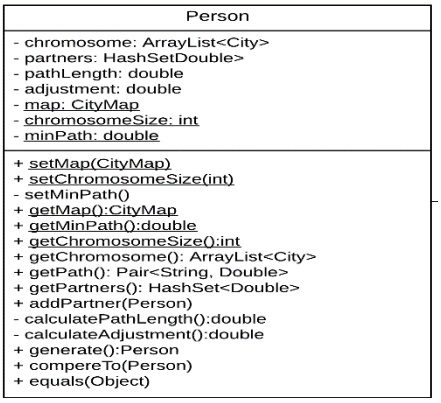

Figure 4. Class diagram of the individual

Evolution occurs in each generation. Generation is an imitation of the life cycle wherein every generation has a population of individuals and the breed within that population. Only the individuals with the highest adaptivity score are kept alive [14].

Generations must be limited by their size and the number of individuals who have the right to breed. These parameters can be changed to observe the change in evolution.

For evolution to occur, we need to create our starting generation and other generations based on previous ones.

Generating the starting generation

We need to implement a method for removing individuals whose adaptivity score is too low because not every individual can reproduce or be moved to the next generation.

And an option to get the best individual in the generation has been added to collect additional static data (Fig. 5).

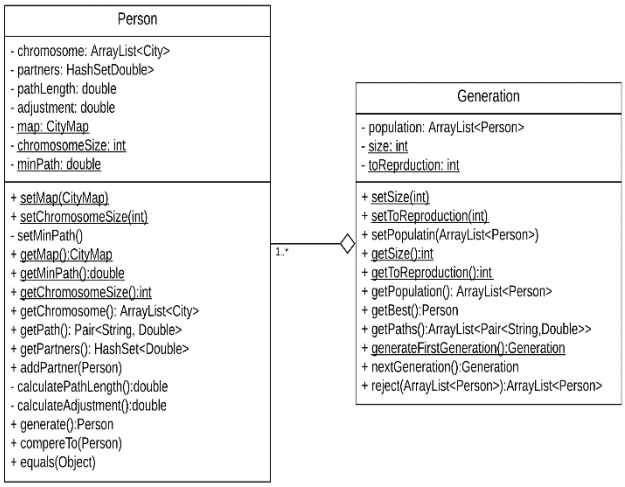

Figure 5. Class diagram of the individual

To generalize the algorithm and data, a Calculation class has been created in which the following data is stored:

- Size of the population.
- Number of generations.
- Percentage of individuals who can reproduce.
- Error($\boldsymbol{\varepsilon}$) that determines the stopping of the algorithm.
- Reproduction type.

The two main methods are shown below. The first one launches the algorithm with the required parameters, and the second one saves the resulting data into a file.

Implementation of the cross-breeding and mutation methods

Cross-breeding and mutation methods are implemented inside an enumerative class where each element represents one method of cross-breeding and mutation.

Also, the ability to acquire descendants from the breeding individuals has been added to that class.

The generalized representation of this class (Fig. 6):

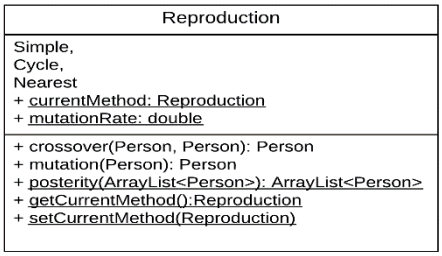

Figure 6. Class diagram of the individual

Inside the first method, “crossover”, everything takes place in two steps. The first step is the combination of parent chromosomes, and the second step is the formatting of the resulting chromosome to make sure it fits the standard model.

Mutation, on the other hand, is realized as a swap of two random genes.

Inside the second method, “mutation”, everything is a bit more complicated. First of all, an attempt is made to copy parents’ transitions, and if at some point it becomes impossible to do so, then all the cities which were not visited are added to it.

The third method, “posterity”, is similar to the second one but slightly changed. The shortest possible parent transition is selected.

Because both the second and the third methods have a possibility where the parents’ steps cannot be repeated, we need to add all the cities which were not visited by those chromosomes.

IV. RESULTS

We will conduct a study on equal terms for each one of the methods. For this, we shall use the same map and the starting generation (Fig. 7).

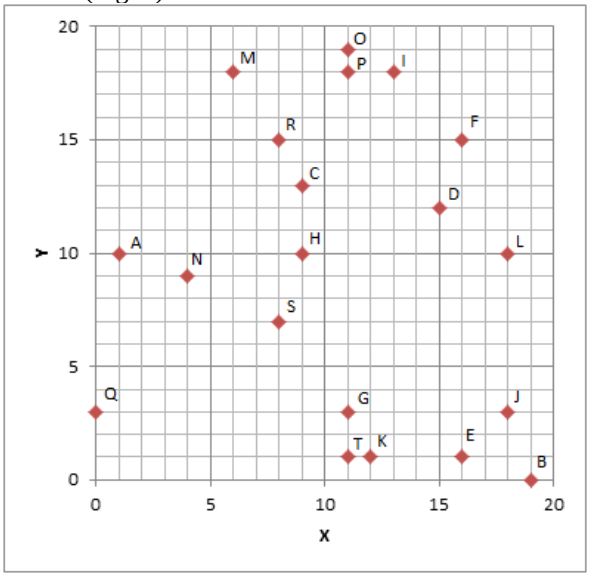

Figure 7. View of the cities on the coordinate plane

Let us generate a map with 20 cities (Table 1).

Table 1. Coordinates

Name	x	y
A	1	10
B	19	0
C	9	13
D	15	12
E	16	1
F	16	15
G	11	3
H	9	10
I	13	18
J	18	3
K	12	1
L	18	10
M	6	18
N	4	9
O	11	19
P	11	18
Q	0	3
R	8	15
S	8	7
T	11	1

Analyzing the efficiency of the cross-breeding method

We conduct a study on 50 generations, with each size being 100 individuals and the possibility to reproduce in 90% of the population.

In the first method, "crossover", the most efficient route is "N-A-S-B-E-J-G-T-K-L-D-F-O-M-R-C-P-I-H-Q" (Fig. 8). Where its length is 109.67769964641892, and the runtime is 5501 ms, 110 ms per iteration.

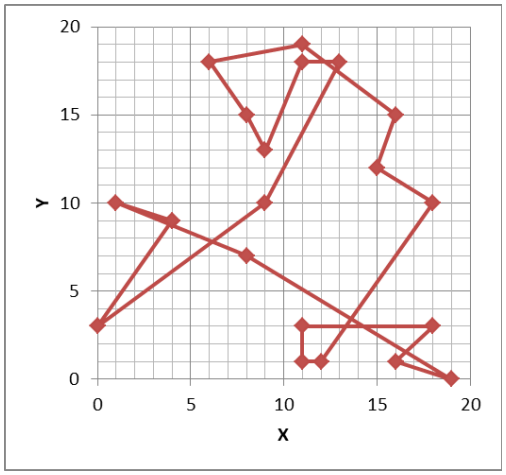

Figure 8. View of the first strategy (Simple)

In the second method, “mutation”, the most efficient route is “C-A-H-R-N-Q-S-T-K-J-B-E-G-L-F-I-D-M-O-P” (Fig. 9). Where its length is 118.9046735493942, and the runtime is 3865 ms, 77 ms per iteration.

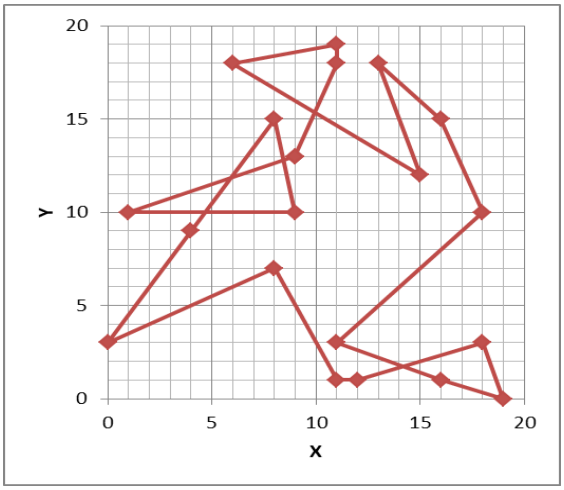

Figure 9. View of the second strategy (Cycle)

In the third method, the most efficient route is "H-C-R-M-P-O-I-F-D-L-J-B-E-K-T-G-S-Q-A-N" (Fig. 10). Where its length is 77.69, and the runtime is 2341 ms, 46 ms per iteration.

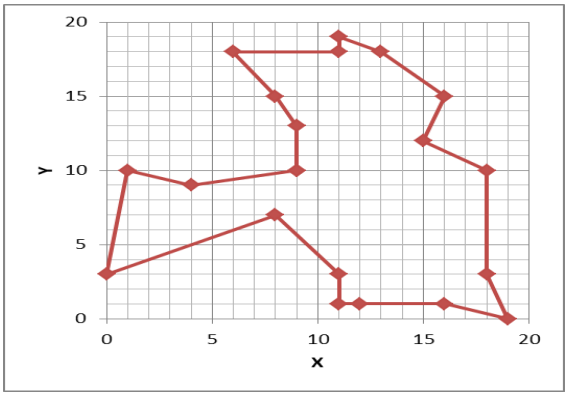

Figure 10. View of the third strategy (Nearest)

As we can see, the third algorithm, “posterity”, has the best runtime and gives us the shortest final route. The second algorithm is average in both runtime and the final route. And the first one has the worst runtime and returns mediocre results.

An analysis of the approximation to the optimal solution

As we can see from the chart (Fig. 11), the third method gives us the best results, and the effects of simple and cycle methods are quite similar.

The nearest algorithm has the most significant decline, and it needs fewer generations to achieve the optimal solution.

The cycle gave us some interesting results (Fig. 11). It has a jump-like approximation, and it happens much less often than in other algorithms. And as a result, the best-fitted individual survives for much longer than other simple and nearest algorithms.

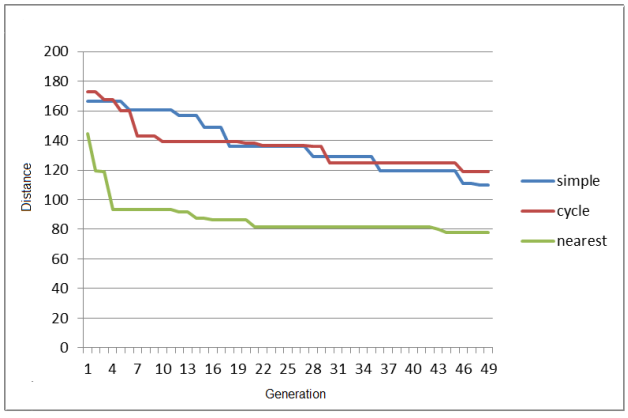

Figure 11. Chart of the speed of approximation to the optimal solution

V. CONCLUSION

As we can see, genetic algorithms are an excellent way to find the best possible solution. They have a reasonably short execution when one iteration takes about 46-110 ms in 20 cities. We considered three methods of crossing, and the third of them was the most effective. This is because the third has one feature from the nearest neighbour, given that each child has better adaptability than its parents.

The paper proposes three methods of intersection. After all, in classical genetic algorithms, crossover occurs by simply linking different halves of both parents’ chromosomes. Therefore, it is not suitable for solving our problem, because with its help we will get to the same city twice and will not visit other cities. A similar situation will occur when using a specific mutation. Therefore, if you complicate crossover and mutation, you can avoid this situation.

To do this, use three strategies. The first is to replace the recurring city with the missing one. At the same time, this method assumes the fact that each possible offspring may have several options. The disadvantage of this method lies in the fact that many iterations need to be performed for one intersection to find repetitions.

The second strategy is the bonding of parental genes to form a Hamiltonian cycle. Based on the second method, you can create an even more efficient crossover.

The third strategy is choosing the most efficient move, speeding up the algorithm, and reducing the number of required generations. A side effect of this strategy is that the offspring can become a copy of the father. However, this can be solved by mutating the offspring or removing it from the general genetic population. The third strategy has a feature of the method of the nearest neighbour, which, in turn, makes each offspring more effective than his father.

Therefore, the article describes various aspects of the study of genetic algorithms and mechanisms for their optimization for the problem of the seller-travelled.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[2] X. Chen, P. Zhang, G. Du and F. Li, “Ant colony optimization based memetic algorithm to solve bi-objective multiple traveling salesmen problem for multi-robot systems,” IEEE Access, vol. 6, pp. 21745–21757, 2018. https://doi.org/10.1109/ACCESS.2018.2828499.

[3] E. Hadjiconstantinou and N. Christofides, “An exact algorithm for general, orthogonal, two-dimensional knapsack problems,” *European Journal of Operational Research*, vol. 83, pp. 39-56, 1995. https://doi.org/10.1109/ACCESS.2018.2828499.

[4] O. E. Semenkina, E. A. Popov, O. E. Semenkina. “Self-configuring evolutionary algorithms for travelling salesman problem,” *Journal of Siberian State Aerospace University named after academician M. F. Reshetnev*, vol. 4, no. 50, pp. 134-139, 2013.

[5] R. D. Tsai, E. M. Malstrom and H. D. Meeks, “A two-dimensional palletizing procedure for warehouse loading operations,” *IIE Transactions*, vol. 20, pp. 418–425, 1988. https://doi.org/10.1080/07408178808966200.

[6] L. S. Buriol, P. Moscato, P. França, “A new memetic algorithm for the asymmetric traveling salesman problem,” *Journal of Heuristics*, vol. 10, pp. 483–506, 2004. https://doi.org/10.1023/B:HEUR.0000045321.59202.52.

[7] M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge USA, London UK: MIT Press, 1999.

[8] N. Boyko, A. Pytel, "Application of genetic algorithms for optimization of salesman's tasks and their modeling by sequential selection," *Proceedings of the 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS 2021)*, vol. I: Main Conference, Lviv, Ukraine, April 22-23, 2021, pp. 969-981.

[9] N. Boyko, "A look through methods of intellectual data analysis and their applying in informational systems," *XIth International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT)*, September, 2016, pp. 183-185. https://doi.org/10.1109/STC-CSIT.2016.7589901.

[10] D. Whitley, *A Genetic Algorithm Tutorial*, 1993. https://doi.org/10.1007/BF00175354.

[11] J. Majumdar, A. K. Bhunia, "Genetic algorithm for asymmetric traveling salesman problem with imprecise travel times," *Journal of Computational and Applied Mathematics*, Elsevier, vol. 235, issue 9, pp. 3063-3078, 2011. https://doi.org/10.1016/j.cam.2010.12.027.

[12] N. Allahverdi, N. Allahverdi, "Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms," *Expert Systems with Applications*, vol. 38, issue 3, pp. 1313-1320, 2011. https://doi.org/10.1016/j.eswa.2010.07.006.

[13] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 63, pp. 337–370, 1996. https://doi.org/10.1007/BF02125403.

[14] K.K. Lai and J.W.M. Chan, "Developing a simulated annealing algorithm for the cutting stock problem," *Computers & Industrial Engineering*, vol. 32, pp. 115-127, 1997. https://doi.org/10.1016/S0360-8352(96)00205-7.

[15] D. Korpyljov, T. Sviridova, S. Tkachenko, "Using of genetic algorithms in design of hybrid integrated circuits," *Proceedings of the IXth International Conference on "The Experience of Designing and Application of CAD Systems in Microelectronics" CADSM 2007*, Polyana, Ukraine, 2007, pp. 302. https://doi.org/10.1109/CADSM.2007.4297557.

[16] A. Shabalov, E. Semenkin, P. Galushin, "Automatized design application of intelligent information technologies for data mining problems," *Proceedings of the 7th Joint IEEE International Conference on Natural Computation & The 8th International Conference on Fuzzy Systems and Knowledge Discovery*, Shanghai, China, 2011, pp. 2659–2662. https://doi.org/10.1109/FSKD.2011.6020026.

[17] Y. Hrytsyshyn, R. Kryvyy, S. Tkatchenko, "Genetic programming for solving cutting problem," *Proceedings of the IXth International Conference on The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2007*, Polyana, Ukraine, 2007, pp. 280-282. https://doi.org/10.1109/CADSM.2007.4297550.

[18] E. Semenkin, M. Semenkina, "Self configuring genetic algorithm with modified uniform crossover operator," *Advances in Swarm Intelligence*, ICSI 2012, Part 1, LNCS 7331, Springer, Heidelberg, 2012, pp. 414-421. https://doi.org/10.1007/978-3-642-30976-2_50.

[19] M. Gen, R. Cheng, *Genetic Algorithms and Engineering design*, John Wiley & Sons, 1997, 352 p. https://doi.org/10.1002/9780470172254.

[20] J. Gaber, M. Bakhouya. "An immune inspired-based optimization algorithm: application to the traveling salesman problem," *Advanced Modeling and Optimization*, vol. 9, issue 1, pp. 105–116, 2007.

[21] L. Grady, E. L. Schwartz, "Isoperimetric graph partitioning for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, issue 3, pp. 469-475, 2006. https://doi.org/10.1109/TPAMI.2006.57.

[22] P. Larra Naga, C.M.H. Kuijpers, R.H. Murga, I. Inza and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, Kluwer Academic Publishers, Printed in the Netherlands, vol. 13, issue 2, pp. 129–170, 1999. https://doi.org/10.1023/A:1006529012972.

[23] S. Rana, *Examining the Role of Local Optima and Schema Processing in Genetic Search*, 1999.

NATALIYA BOYKO, PhD, an Associate Professor of the Artificial Intelligent Systems Department of Lviv Polytechnic National University. Scientific interests: machine learning, data visualization, intellectual data analysis, system analysis.

ANDRIY PYTEL is a student of the Artificial Intelligent Systems Department of Lviv Polytechnic National University.