Application of Genetic Algorithms for Optimization of Salesman's Tasks and Their Modeling by Sequential Selection

Nataliya Boyko and Andriy Pytel

Lviv Polytechnic National University, Profesorska Street 1, Lviv, 79013, Ukraine

Abstract

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

Keywords 1

Genetic Algorithms, Cross-Breeding, Crossover, Mutation, Generations, Individuals, Selection, Evolution, Travelling Salesman Problem, City, Route

1. Introduction

One of the common problems in programming is the traveling salesman problem. There are many various methods that can be used to solve it, but the one we are going to take a look are the genetic algorithms. The aim of this study is to the most efficient application of genetic algorithms in the travelling salesman problem [3, 19].

Lately artificial intelligence has been becoming more and more popular, but at the same time a stereotype has been formed that AI is only is based solely on the neural networks even though a neural network is only one of the numerous directions of artificial intelligence. The aim of this paper is to bring attention to other directions of AI, such as genetic algorithms [5, 15].

Study the process of solving the travelling salesman problem (TSP) via genetic algorithms (GA) and take a look at the problems of this method.

The origins of the travelling salesman problem are unclear. A handbook for travelling salesmen from 1832 mentions the problem and includes example tours through Germany and Switzerland, but contains no mathematical treatment [7-8, 16].

It was first considered mathematically in the 1930s by Merrill M. Flood who was looking to solve a school bus routing problem. Hassler Whitney at Princeton University introduced the name travelling salesman problem soon after [11, 12].

There are many methods of solving this problem. Some of them give exact results, others only approximate. One of the more interesting methods is the methods of route optimization with the use of genetic algorithms [1, 14, 22].

COLINS-2021: 5th International Conference on Computational Linguistics and Intelligent Systems, April 22–23, 2021, Kharkiv, Ukraine
EMAIL: nataliya.i.boyko@lpnu.ua (N. Boyko); pytelandriy@gmail.com (A. Pytel)
ORCID: 0000-0002-6962-9363 (N. Boyko); 0000-0002-6187-740X (A. Pytel)

First experiments which involved simulated evolution were conducted by Nils Aall Barricelli in 1945. Later they were also conducted by Nils Aall Barricelli, Ingo Rechenberg and Hans-Paul Schwefel. Thanks to them artificial evolution became a well-known optimization method [3].

2. Theoretical basis

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

The travelling salesman problem can be presented as a [6, 15, 23]:

- 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 direction has no role.

There are two main groups of methods for solving the travelling salesman problem which can be combined [5,10, 17]:

- Precise — they find the precise optimal solution to the problem, but takes 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 which do not guarantee a precise solution, but only the best possible approximation with the given amount of time(iterations) [9, 18].

3. Genetic algorithms

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

Genetic algorithms can be broken down into the following steps [4, 21]:

1. Creation of the base population.
2. Repeat 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 on the Figure 1.

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. I 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 will be stored e.g. ABCDE, BADCE, BDCEA.

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

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

Now we can commence our selection. We do that by selecting the best fitted individuals for their cross-breeding and/or mutation [8, 20].

In order 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 the sexual reproduction. In order to create a new individual (child) 2 individuals are required which will be called parent in relation to the child. To inherit parent

traits child chromosomes are formed in a certain way by combining parent genes. This process is called a crossover.

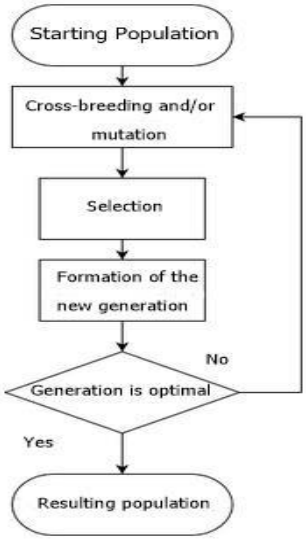


Figure 1: Stages of the genetic algorithm

In case when both mutation and cross-breeding have been performed in order to avoid a situation in which the population did not improve or got stuck on a certain iteration some if not all individuals are mutated. In other words one or some genes in a chromosome are replaced. Even though this process of mutation is not mandatory as mutations can both improve the rate of approximation to the result and slow it down [7, 21].

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

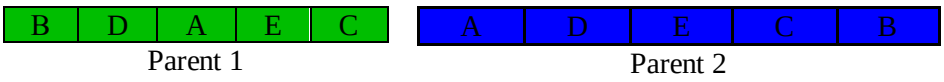
If the generation is not adapted enough we repeat processes of rating, selecting, cross-breeding / mutation and forming the new generation.

If the new generation is adapted enough, it can be considered to be resulting. In our case the route is as short as possible and any improvements are either insignificant or are impossible [1-3, 13].

4. Analytical part

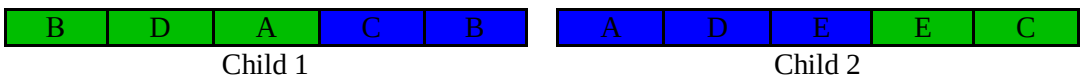
The complexity of using this method is that isn't possible to use a regular crossover or a mutation.

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



Parent 1 Parent 2

Then our children will have the following appearance.



Child 1 Child 2

As we can see the regular crossover is not applicable 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 regular mutation.

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

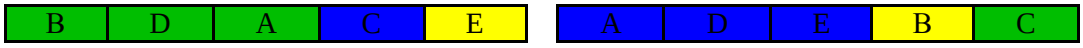
4.1. The first strategy (Simple)

As was mentioned before, in order to avoid getting to the same city twice merely differently combining halves of parent chromosomes will not be sufficient. 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 the fact 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 amount of iterations which are required to find the repeats which are needed to be competed for this cross-breeding. As a result, this method is quite inefficient in terms of runtime.

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

4.2. The second strategy (Cycle)

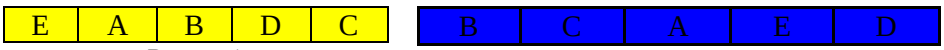
The previous strategy was based on a basic crossover which had to be improved so that children could form a Hamiltonian cycle which in term would create a large quantity of children variations.

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

The idea of this strategy is that the combination of parent genes should immediately form a Hamiltonian cycle [5-7, 16].

In addition to make sure that the child chromosome would not repeat its parent it necessary to limit the length 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

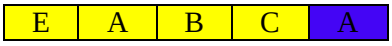
And we can copy up to three genes in one parent in a row.

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). As we can see, it is in the first place. 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 next 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 and moving to the right has a higher priority, so we add C.

After that, we are going to be moving through the parent in the same direction until we hit a city to which we have been to 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 will get:



4.3. 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 and if the most efficient one 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 and at the same time will increase the time designated for computing in a single generation.

In order to make sure that the complexity of the algorithm will not increase the restriction regarding copying the fathers' 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:

Parent 1

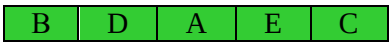
Parent 2

In this case we are not just going to the first available city, but rather we choose and optimal path. In the beginning a starting gene is randomly chosen, so the simplicity sake in this example the first parent gene will be chosen, let us 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 let us assume that C is the shortest. And as a result we get more children which are more efficient than their parents.

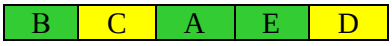
5. Mutation

In order to simplify the work we will take the method of mutation from the first method i.e. the process of mutation goes as it would go for a regular genetic algorithm. Then we will correct the cities which are missing. As a result, we will just swap the places of two random genes.

The chromosome before mutation:



And after mutation:



In order to improve the algorithm, we could also look for the most efficient swap, but that would only waste resources it will amount to little to no optimization gain.

6. Selection of individuals for reproduction

The next problem that occurs is the selection of individuals which will be bred and how they will be bred. The simplest and possibly the most efficient one method is sorting the individuals according to some coefficient and pairwise crossing of the most adapted individuals [12, 13].

That fact that crossing the same parents will yield the same individual. In order to avoid breeding the same pairs of parents each individual will have its list of partners with whom he created offspring and block its ability to breed with these individuals.

Development of the software solution

The structure of the software solution can be separated into three parts [2]:

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

Structural specifics of the travelling salesman problem.

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

In order to be able to consider the graphic formulation of the route cities are represented as points with their names on the coordinate plane.

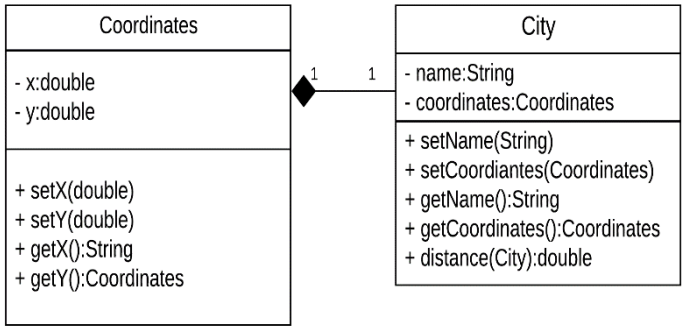


Figure 2: UML diagram of the city class

When selecting the structure to hold the data it is important to take into account that connections between the cities are more important than the cities themselves (Figure 2) and that it is wise to store the calculated distances and retrieve them rather than constantly recomputing them as it will create an excessive load on the program. So for this purpose we will realize a sort of map which will store all the cities and distances between them. To ease the access to data inside the said map the list of cities and the list of distances between them will be presented as hash-maps. Cities will be accessed through their name and the distances will be accessed by using a key which is composed out of the names of the cities distance between which we are looking for [10].

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

Also, in order to not repeat the operation of 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 (Figure 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 one of the possible routes and the adaptivity score is the ratio of the length of the shortest known route to the length of the current individual [7].

In order to determine the best adapted individuals, we need to be able to compare them by their adaptivity score and for this a “Comparable” will be used.

The first generation will be generated with random routes.

The process of generating individuals

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 (Figure 4) [8].

Evolution occurs in each generation. Generation is an imitation of the life cycle where in every generation there is a population of individuals and within that population that population the breed. During the creation of the next generation only the individuals with the highest adaptivity score are kept alive [6].

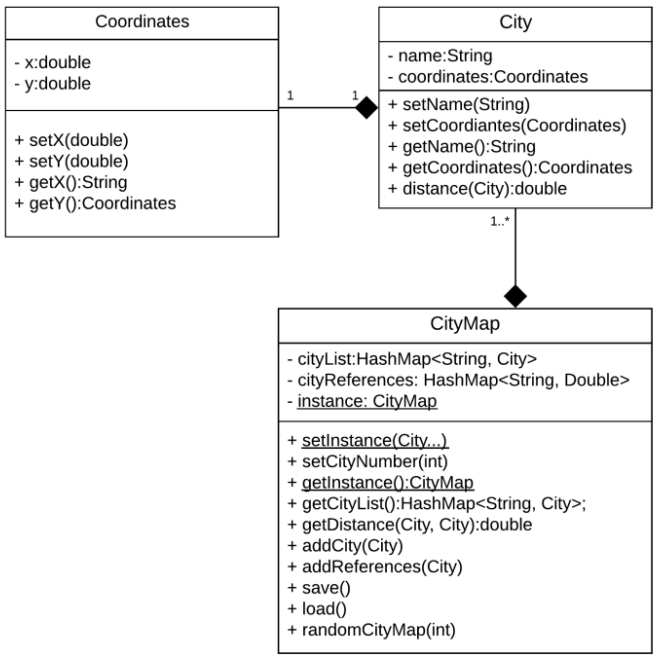


Figure 3: UML maps and their dependencies

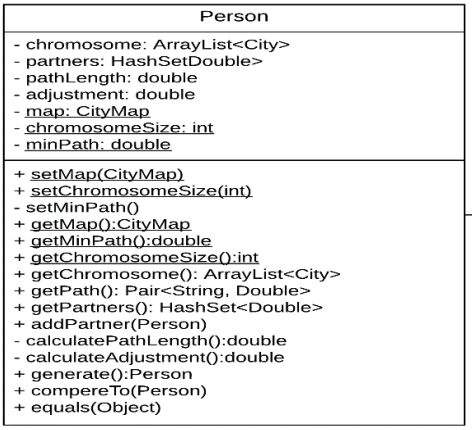


Figure 4: Class diagram of the individual

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

In order for the evolution to occur we need to be able to create our starting generation and other generations based on previous ones.

Generating the starting generation

We need to implement a method for killing individuals whose adaptivity score is too low considering the fact that not every individual has the ability to reproduce or to be moved to the next generation.

And an option to get the best individual in the generation has been added in order to allow us to gather additional static data (Figure 5).

In order 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 which can reproduce.
- Error (ε) that determines the stopping of the algorithm.
- Reproduction type.

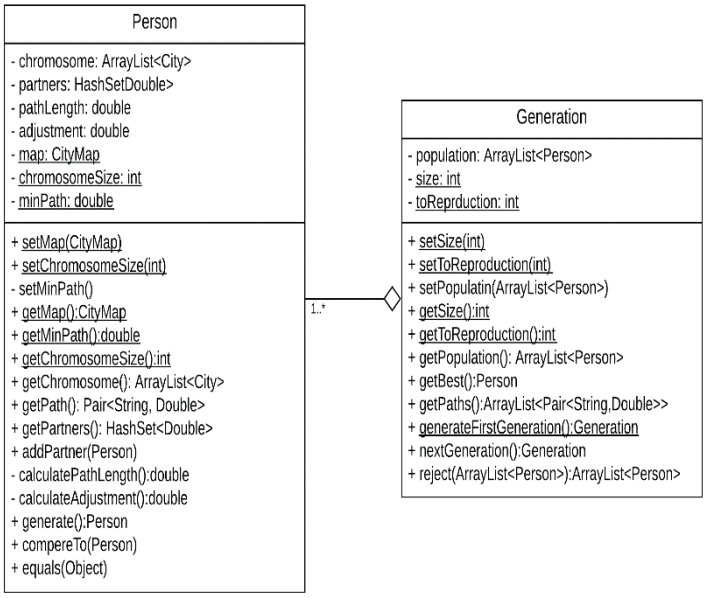


Figure 5: Class diagram of the individual

The two main methods are shown below. The first one launches the algorithm with required parameters and the second one save 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 of acquiring descendants from the breeding individuals has been added in that class.

The generalized representation of this class (Figure 6).

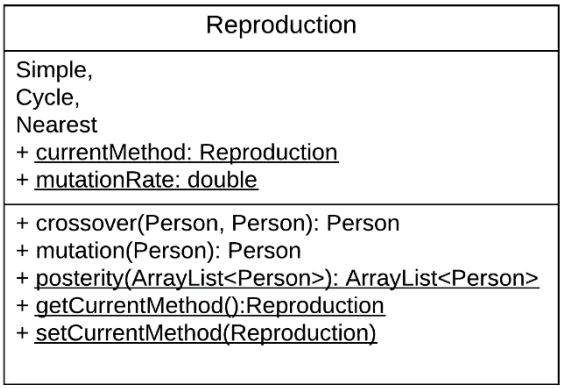


Figure 6: Class diagram of the reproduction implementation

Inside the first method everything takes place in two steps. The first one is the combination of parent chromosomes and the second one is the formatting of the resulting chromosome to make sure it fits the standard model. This method involves adding two sequences of cities and removing repetitions from them.

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

Inside the second method 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 is similar to the second one but with a slight change. The shortest possible parent transition is selected.

7. 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 (Figure 7).

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

Table 1
Coordinates

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

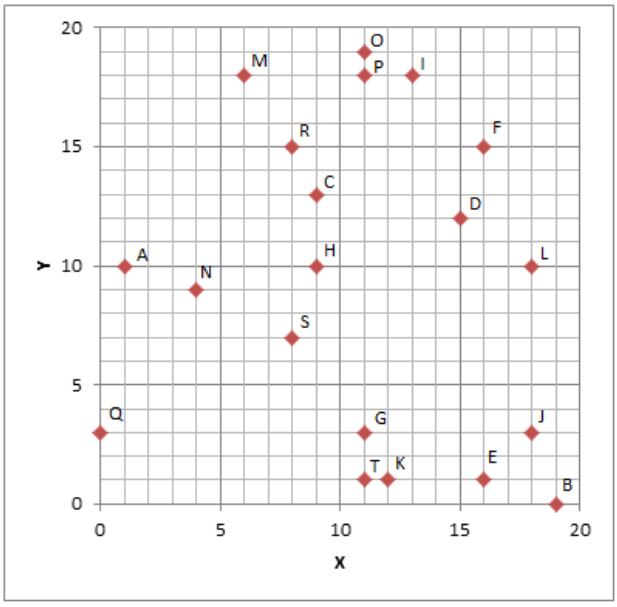


Figure 7: View of the cities on the coordinate plane

Analyzing the efficiency of the cross-breeding method

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

In the first method 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" (Figure 8). Where its length is 109.67769964641892 and the runtime is 5501 ms, 110 ms per iteration.

In the second method 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” (Figure 9).

Where its length is 118.9046735493942 and the runtime is 3865 ms, 77 ms per iteration.

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" (Figure 10). Where its length is 77.68934906917778 and the runtime is 2341 ms, 46 ms per iteration.

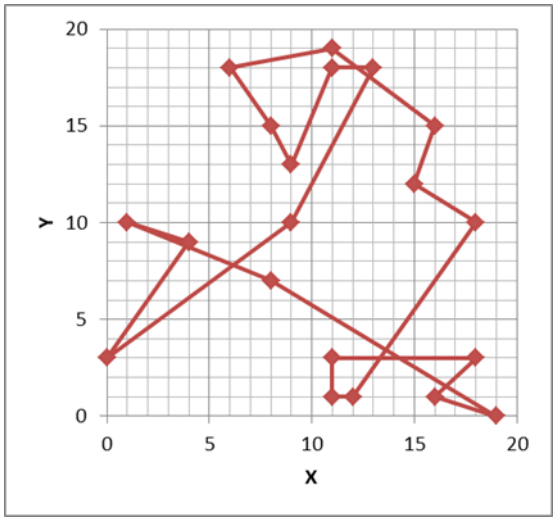


Figure 8: View of the first method

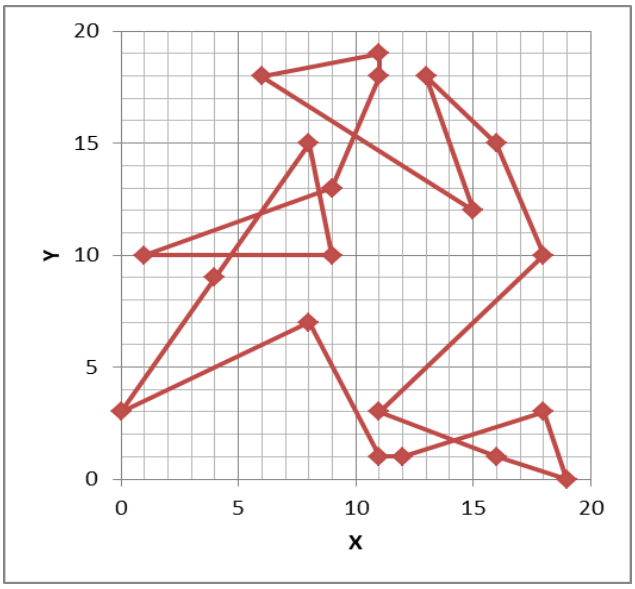


Figure 9: View of the second method

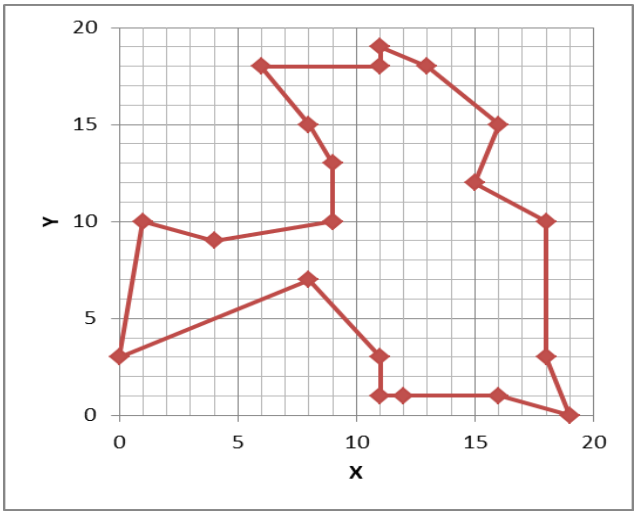


Figure 10: View of the third method

As we can see the third algorithm 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 third 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 the third method gives us the best results and the results of simple and cycle methods are actually quite similar (Figure 11).

The nearest algorithm has the largest decline and because of that it needs fewer generations to achieve the optimal solution.

Cycle gave us some interesting results (Figure 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 compared to other simple and nearest algorithms.

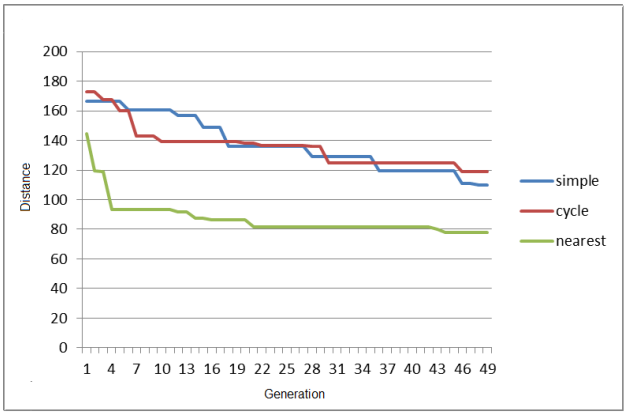


Figure 11: Chart of the speed of approximation to optimal solution

8. Conclusion

In the article three methods of crossing are proposed. After all, in classical genetic algorithms, the crossover occurs through the simple bonding of different halves of the chromosomes of both parents. Therefore, it is not suitable for solving our task, because using it we will get to the same city twice and will not visit other cities. A similar situation will occur when using a normal mutation. Therefore, if you complicate the crossover and mutation, you can avoid this situation.

To do this, use three strategies. The first is to replace a recurring city with a missing one. At the same time, this method entails the fact that each possible offspring may have several options. The disadvantage of this method is the large number of iterations to find repetitions to be performed for one crossing.

The second strategy is the gluing 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 to choose the most efficient move, to speed up the algorithm and to reduce 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 either by mutating the offspring or removing it from the general genetic population. The third strategy has a feature of the method of the nearest neighbor, which in turn makes each offspring more effective than his father.

As we can see genetic algorithms are a nice way of locating the best possible solution. They have a quite short runtime where one iteration takes about 46-110 ms in 20 cities. We took a look at three methods of cross-breeding and from them the most efficient one was the third one. This is thanks to the fact that the third one has one special feature from the nearest neighbor which in terms makes every child have a better adaptivity score than its parents.

9. References

[1] M. Albayrak, N. Allahverdy Development a new mutation operator to solve the Traveling Salesman Problem by aid of genetic algorithms. Expert Systems with Applications, 38(3), 2011, pp. 1313–1320.

[2] J.-Y. Potvin, X. Ying, I. Benyahia, Vehicle routing and scheduling with dynamic travel times, Computers and Operations Research 33, 2006, pp. 1129-1137.

[3] J.-Y. Potvin, Genetic algorithms for the traveling salesman problem, Annals of Operations Research 63, 1996, pp. 339-370.

[4] J.-F. Bérubé, M. Gendreau and J.-Y. Potvin, An exact epsilon-constraint method for bi-objective combinatorial optimization problems: Application to the Traveling Salesman Problem with Profits, European Journal of Operational Research 194, 2009, pp. 39-50.

[5] M. Tagmouti, M. Gendreau, J.-Y. Potvin, Arc routing problems with time-dependent service costs, European Journal of Operational Research 181, 2007, pp. 30-39.

[6] S. Ichoua, M. Gendreau, J.-Y. Potvin, Exploiting knowledge about future demands for real-time vehicle dispatching, Transportation Science 40, 2006, pp. 211-225.

[7] M. Gendreau, F. Guertin, J.-Y. Potvin and R. Séguin, Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries, Transportation Research Part C 14, 2006, pp. 157-174.

[8] M. Gendreau, J.-Y. Potvin, A. Smires, P. Soriano, Multi-period capacity expansion for a local access telecommunications network, European Journal of Operational Research 172, 2006, pp. 1051-1066.

[9] S. Ichoua, M. Gendreau, J.-Y. Potvin, Vehicle dispatching with time-dependent travel times, European Journal of Operational Research 144, 2003, pp. 379-396.

[10] B. Gendron, J.-Y. Potvin, P. Soriano, A tabu search with slope scaling for the multicommodity capacitated location problem with balancing requirements, Annals of Operations Research 122, 2003, pp. 193-217.

[11] D. Berger, B. Gendron, J.-Y. Potvin, S. Raghavan and P. Soriano, Tabu Search for a Network Loading Problem with Multiple Facilities, Journal of Heuristics 6, 2000, pp. 253-267.

[12] N. Kunanets, O. Vasiuta, N. Boiko, Advanced Technologies of Big Data Research in Distributed Information Systems, in: Proceedings of the 14th International conference "Computer sciences and Information technologies" (CSIT 2019), September 17-20, 2019, Lviv, Ukraine, pp. 71-76.

[13] Y. Hrytsyshyn, R. Kryvyy, S. Tkatchenko, Genetic Programming For Solving Cutting Problem, in: 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.

[14] D. Korpyljov, T. Sviridova, S. Tkachenko, Using of genetic algorithms in design of Hybrid Integrated Circuits, in: Proceedings of the IXth International Conference on “The Experience of Designing and Application of CAD Systems in Microelectronics” CADSM 2007, Polyana, Ukraine, 2007, 302 p.

[15] N. Boyko, A look trough methods of intellectual data analysis and their applying in informational systems, in: XI-th International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT), September, 2016, pp. 183-185.

[16] 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, 2011, pp. 3063-3078.

[17] 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(50), 2013, pp. 134-139.

[18] N. Boyko, A. Bronetskyi, N. Shakhovska, Application of Artificial Intelligence Algorithms for Image Processing, in: CEUR. Workshop Proceedings of the 8th International Conference on “Mathematics. Information Technologies. Education”, MoMLeT&DS-2019, Vol. 2386 , Shatsk, Ukraine, June 2-4, 2019, pp. 194-211.

[19] A. Shabalov, E. Semenkin, P. Galushin, Automatized Design Application Of Intelligent Information Technologies for Data Mining Problems, in: The 7th Inter-national Conference on Natural Computation & The 8th International Conference on Fuzzy Systems and Knowledge Discovery, Shanghai, China, 2011, pp. 2659–2662.
[20] E. Semenkin, M. Semenkina, Self configuring genetic algorithm with modified uniform crossover operator, in: Advances in Swarm Intelligence, ICSI 2012, Part 1, LNCS 7331, Springer, Heidelberg, 2012, pp. 414.–421.
[21] 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, 2018. Digital Object Identifier 10.1109/ACCESS.2018.2828499
[22] L. Grady, E. L.Schwartz, Isoperimetric Graph partitioning for Image segmentation, in: IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.28(3), 2006, pp.469-475.
[23] J. Gaber, M. Bakhouya, An Immune Inspired-based Optimization Algorithm: Application to the Traveling Salesman Problem, in: Advanced Modeling and Optimization, Vol. 9(1), 2007, pp. 105–116.