Understanding Genetic Algorithms and Their Applications in Finding Solutions to the Travelling Salesman Problem

Yaroslav Mikhaylik

COMP3190 Project, University of Manitoba

mikhayly@myumanitoba.ca

December 1st, 2020

Abstract

This paper focuses on investigating how and why genetic algorithms (GAs) work, as well as their application to a particular travelling salesman problem (TSP). The final goal is to show that genetic algorithms could be used to find near-optimal solutions to problems that are otherwise close-to-impossible to bruteforce solutions to. The following work consists of two major parts: the theoretical foundation and practical application. The former one is required for us to gather enough of basic material without necessarily restricting us to bounds of any specific problem. But for the latter part we should be able to apply the gained knowledge to come up with multiple

solutions to the TSP. Finally, we will document our findings in a conclusion to this paper.

1. Theory of genetic algorithms

1.1. Evolutionary algorithms

To introduce GAs we first need to explore the concept of evolutionary algorithms (EAs). The EAs are based off real life Darwinian evolutionary processes which simply rely on the fact that the strongest (the most fit) individual always survives, and as such the next generation becomes stronger and healthier than the previous one [Hussain et al., 2017]. One of the outstanding subsets of EAs are the GAs, which very closely simulate the survival-of-the-fittest process, however, it should be acknowledged that there do exist other EAs besides GAs, ie: genetic programming, evolution strategy, neuroevolution and more [Larrañaga et al., 1999], though the focus of this paper is entirely on the GAs .

1.2. Genetic algorithms

Genetic algorithms are probabilistic search algorithms which simulate natural evolution in an attempt to solve an optimization problem [Abdoun et al., 2012]. First, the initial population of some individuals is chosen, and the quality of each individual in the population is assessed by some fitness function [Larrañaga et al., 1999]. Next, in every following iteration parents are selected from the population by some selection method based on their fitness [Larrañaga et al., 1999]. Sometimes individuals are also called chromosomes, since by their nature they have some information that describes them in the present time, and how they could be improved in the future [Hussain et al., 2017]. It should be noted here that every next iteration closely resembled the word *generation*, so

we will be using the latter word hereafter. The selected individuals become parents, and via breeding they produce offspring in a way that combines the best characteristics of parents [Larrañaga et al., 1999]. The offspring are then either added to the existing population or form a different generation independent of their parents [Hussain et al., 2017]. For all newly created offspring there is some chance that they will mutate: in other words, they will somehow be distinct from their parents [Larrañaga et al., 1999].

The respective names of the operators which define the child production process and the mutation process are the crossover operator and the mutation operator [Larrañaga et al., 1999]. The choice of a correct crossover operator is guaranteed to increase the quality of each generation [Larrañaga et al., 1999]. Mutation is required in GAs to help the algorithm in avoiding local optima [Larrañaga et al., 1999]. We choose adequate crossover and mutation operators with the intent to increase the probability that the GA converges to a near-optimal solution in a reasonable amount of time.

GAs must be balanced between exploitation and exploration [Jebari, 2013]. For example, for each selection method there exists a pressure factor to create a competition between the individuals to be selected for breeding [Miller and Goldberg, 1995]. “In the case of selection, a strong selection pressure may cause the algorithm to converge to a local optimum, while a low selection pressure may cause the GA to produce random results that differ from one generation to the next” [Jebari, 2013]. As such, we can conclude that the selection operator is the most important parameter that influences the performances of any GA [Jebari, 2013]. Though, as mentioned above, the mutation also plays a huge role in preventing the stagnation of each generation, as with

each offspring it introduces some variation to avoid possible deadends of local optima [Larrañaga et al., 1999].

2. Practical application of genetic algorithms

2.1. Travelling salesman problem in the context of Saint-Petersburg's subway system

One of the many areas of application of GAs is in the solving of TSP. For the purposes of this document we will introduce an example of TSP, on which we will later use GAs to find an optimal route that takes the least amount of time to traverse.

In this example we are focusing on a subway system of Saint-Petersburg city which consists of 71 active stations which will play a role of cities in TSP (see Attachment 5.1 for the subway map). We have a goal of traversing an entire subway system in the least possible amount of time, so we take time to be the metric of how good the final route is. As a part of this goal, we would like to step out of the train on every station's platform, so there is a chance we would have to miss a train on some stations if we are too slow to get back on to the same train. Moreover, at some station-junctions we are required to walk between stations, and in the worst case scenario we arrive at the station right when the previous train is leaving, so we need to wait an additional interval before a new train arrives. This gives us two primary methods of transportation: trains and walks. Moreover, this makes time a complex metric that consists of two primary parts: travel interval and travel time. The travel interval is a

measure in minutes of how often some kind of method of transportation is available from one neighbor to the other. The travel time is a measure in minutes of how long that method of transportation is going to take between the two neighbouring stations.

We define a worst-travel-time (WTT) metric, which considers the worst case scenario where we have to await a new train at every station we have exited. See Attachment 5.2 for sample calculations for WTT of parts of a route. It should be noted that travel times at station-junctions are zero, because we consider that walking does not require any additional waiting interval. With this definition in mind, WTT becomes an ideal upper bound for the route's total travel time. In the case where we do not have to wait for any additional intervals (ie: all trains arrive exactly right on time at stations-junctions, and we are able to get back onto the train we have stepped out of before it leaves), then we would get an ideal lower bound for the total travel time. In other words, lower bound for total travel time is the one that has all travel intervals equal to zero.

2.2. Gathering data

The biggest challenge to constructing this TSP is gathering data for the weighted graph. We are specifically interested in travel intervals and travel times between each pair of neighboring stations. Luckily for us, there exists a service called Yandex.Metro that contains such information that could be parsed with some browser automation via Selenium. The downside is that this information is hard-coded and does not represent how train scheduling might change during the time of day. However, this could be

something to consider in the future if we wanted to get better results from a TSP based on a time of day. Moreover, in this model we are assuming that all stations are open from 7am and are closed from 11pm and this broad assumption can also be improved in the future iterations of the given problem. To adjust for such assumptions we have added a support for interval and time coefficients to add some penalty to total travel time calculations.

We store the gathered data in a JSON file that is later used in building a weighted graph where subway stations represent the vertices and travel intervals and travel times represent the edges. Since we want to evaluate how long each route is going to take, we need to be able to compute a total travel time between any of the two stations in the system. For this we are using Dijkstra's algorithm on the constructed weighted graph - since it is proven to be optimal - and add together total travel times for every station in a route.

2.3. Implementing the genetic algorithm

We have mentioned the word route a lot, but to give it a better definition a route is a permutation of stations. In all of the following discussions we are using a path representation for a route, as it is the easiest one to work with when it comes to GAs [Abdoun et al., 2012]. Of course, the goal is to get the perfect permutation that minimizes the travel time. Trivially, routes have a fixed length of 71 stations, so each route is equivalent to the definition of a fixed-length chromosome given in the theoretical section. We also take our fitness for each route to be the reciprocal of the

total travel time, so higher fitness equates to lower travel time. Note that the use of GAs for this problem is justified by its size, since to find a good permutation just by pure brute-force we would need to check 71! permutations, which is clearly unreasonable.

We have implemented several selection methods: tournament selection, random selection, truncation selection and proportion selection [Jebari, 2013]. These were a good starting point at understanding how many ways one can select individuals for mating, but in no way the methods are limited to just these four. The tournament selection was the first choice because of how easy it is to implement and how good the final results were. In tournament selection we repeatedly draw some fixed number of random individuals to compete in a tournament and the individual with highest fitness gets put into the mating pool [Miller and Goldberg, 1995]. The best part about tournament selection is that we can directly affect the selection pressure by just increasing the tournament size, “as the winner from a larger tournament will, on average, have a higher fitness than the winner of a smaller tournament” [Miller and Goldberg, 1995]. In random selection we just randomly choose individuals from a population which has proven to be useless because there was no pressure factor associated with this method to drive evolution [Miller and Goldberg, 1995]. Truncation selection orders all individuals in the population based on their fitness and then selects only the individuals with the highest fitness, however it is less used in practice because of how trivial it is [Jebari, 2013]. Proportion selection is sometimes referred to as a roulette wheel selection, where all individuals are first sorted in order of increasing fitness, and then they are chosen for mating with weighted probability [Jebari, 2013].

The drawback of this method is that very often GA converges to some local optima [Jebari, 2013].

After the mating pool is populated in the selection process we come to the most important part - the crossover method - how these individuals are bred together. The problem that arises in some crossover methods for TSP is that we could get invalid routes, ie: routes with duplicate/missing stations [Larrañaga et al., 1999]. There are several ways to resolve this, for example brute-forcing random algorithms with two parents until one yields valid offspring routes, ignoring the offspring, attempting repair algorithms to fix the offspring and others [Larrañaga et al., 1999]. Since these could get quite complicated, we are opting for an easier method of using crossover methods that are designed for TSPs and avoid the above problems in offspring [Hussain et al., 2017].

For the TSP-compatible crossover methods we have chosen to investigate ordered crossover and cycle crossover. In ordered crossover we cut away parts of a route of one parent and fill remaining spots with remaining parts of the other parent in a way that avoids duplicate/missing parts [Hussain et al., 2017] Moreover, ordered crossover was shown to have higher convergence rate than cycle crossover [Hussain et al., 2017]. In cycle crossover the idea is to walk through the cycles of both parents to generate the offspring, however the drawback of this is that sometimes we can get the same offspring as the parents [Hussain et al., 2017].

And at last we have mutation methods, which were the simplest ones to implement and all of them had drastic effects on the GA's convergence speed. We have explored simple inversion mutation, inversion mutation, swap mutation and scramble

mutation. In reality, it is not hard to come up with your own mutation methods as long as some stops in a route get somehow shuffled [Abdoun et al., 2012]. “The simple inversion mutation operator selects randomly two cut points in the string, and it reverses the substring between these two cut points” [Larrañaga et al., 1999], which has shown to be the most efficient mutation method for the set up problem. The inversion mutation is similar, but it cuts away some part of a route and inserts it somewhere else in a reversed order [Larrañaga et al., 1999]. The swap mutation is sometimes called an exchange mutation, where it randomly selects two points in a route and swaps them [Larrañaga et al., 1999]. The scramble mutation simply randomly shuffles points in some part of a route [Larrañaga et al., 1999]. These mutation methods were a good starting point for the GA, but mutations are by no means limited to just these several methods.

With all of these things in mind we were able to write a python implementation of every method mentioned in this section to go through the process of selecting individuals for a mating pool, for breeding the said individuals, and for mutating the resulting offspring to form a new generation. We then empirically compared to determine which ones converged the fastest to global optima in our set up TSP.

2.4. Results

Finding the best route was done through empirical testing of what selection, crossover and mutation methods converge the fastest to the global optima. As such,

this by no means says that the following methods are the best for all GAs, but they did provide the best results in the current problem.

You can see Attachment 5.3 for the final route with the lowest obtained WTT of 522 minutes. This result was achieved by using tournament selection, ordered crossover and simple inversion mutations. It should be noted that for the execution of the route in the real world we did adjust a small thing about the final route without affecting the WTT: we have put the station with id 10 right after the station with id 14 just to do less walking in station-junctions. To obtain the lower bound of 342 minutes we have set all travel intervals to zero and computed the total travel time of the route without additional intervals.

In the end, we were able to physically ride through this route in exactly 390 minutes with occasional train misses and some exploring on really nice looking train stations. This does fall within the range of expected values, so we can conclude that the GAs successfully generated a route through Saint-Petersburg's subway system that satisfies the required criteria outlined in Section 2.1.

3. Conclusion

In this paper we have built an understanding of why and how GAs work and how they are a subset of EAs, which very closely simulate real-life evolutionary processes. We have further explored how close GAs are to the Darwinian evolution, and why they were a good choice of GAs for attempting to solve optimization problems.

In a practical example we have shown how GAs could be applied in finding solutions to the TSP problem. Moreover, as a further improvement to the subway TSP model we have discussed the potential to more accurately consider the effects of time of day on train schedules. Finally, the approach explained in this paper is general enough so that it could easily be expanded and applied to any other TSP problem.

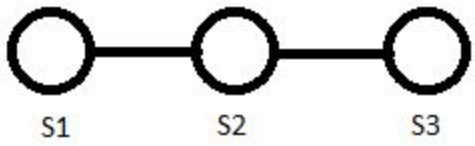
4. Acknowledgments

I extend my appreciation to Professor John Anderson at University of Manitoba for teaching Introduction to AI course and providing me with an opportunity to explore the topic of genetic algorithms as a form of AI by myself.

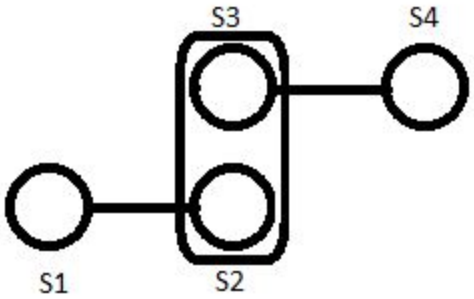
5. Attachments

5.1. Subway map of Saint-Petersburg

5.2. WTT sample calculations



WTT = Interval at S1 + Travel time S1-S2 +
+ Interval at S2 + Travel time S2-S3



WTT = Interval at S1 + Travel time S1-S2 +
+ Interval at S2(0) + Travel time S2-S3 +
+ Interval at S3 + Travel time S3-S4

5.3. The final route

```
Route's wtt=522.0, fitness=0.0019157088122605363
7:00, 2, Parnas
7:06, 2, Prospekt Prosveschenija
7:11, 2, Ozerki
7:17, 2, Udel'naja
7:23, 2, Pionerskaja
7:29, 2, Chernaja rechka
7:36, 2, Petrogradskaja
7:41, 2, Gor'kovskaja
7:50, 2, Sennaja Ploschad'
8:00, 5, Chkalovskaja
8:06, 5, Krestovskij Ostrov
8:16, 5, Komendantskij Prospekt
8:23, 5, Staraja Derevnja
8:34, 5, Sportivnaja
8:40, 5, Admiraltejskaja
8:45, 5, Sadovaja
8:48, 4, Spasskaja
8:53, 2, Nevskij Prospekt
```

8:57, 3, Gostinyj dvor
9:16, 3, Begovaja
9:27, 3, Primorskaja
9:34, 3, Vasileostrovskaja
9:44, 3, Majakovskaja
10:04, 3, Obuhovo
10:11, 3, Rybatskoe
10:21, 3, Proletarskaja
10:28, 3, Lomonosovskaja
10:34, 3, Elizarovskaja
10:41, 3, Ploschad' Aleksandra Nevskogo-1
10:44, 4, Ploschad' Aleksandra Nevskogo-2
10:56, 4, Prospekt Bol'shevikov
11:02, 4, Ulitsa Dybenko
11:11, 4, Ladozhskaja
11:17, 4, Novocherkasskaja
11:25, 4, Ligovskij Prospekt
11:30, 4, Dostoevskaja
11:33, 1, Vladimirskaja
11:38, 1, Pushkinskaja
11:58, 5, Dunajskaja
12:04, 5, Shushary
12:13, 5, Prospekt Slavy
12:18, 5, Mezhdunarodnaja
12:24, 5, Buharestskaja
12:30, 5, Volkovskaja
12:36, 5, Obvodnyj kanal
12:42, 5, Zvenigorodskaja
12:47, 1, Tehnologicheskij institut-1
12:51, 2, Frunzenskaja
12:59, 2, Elektrosila
13:04, 2, Park Pobedy
13:18, 2, Kupchino
13:24, 2, Zvezdnaja
13:32, 2, Moskovskaja
13:42, 2, Moskovskie Vorota
13:50, 2, Tehnologicheskij institut-2
14:01, 1, Kirovskij zavod
14:11, 1, Prospekt Veteranov
14:16, 1, Leninskij prospekt

```
14:22, 1, Avtovo
14:31, 1, Narvskaja
14:37, 1, Baltijskaja
14:48, 1, Ploschad' Vosstanija
14:54, 1, Chernyshevskaja
15:00, 1, Ploschad' Lenina
15:06, 1, Vyborgskaja
15:12, 1, Lesnaja
15:18, 1, Ploschad' Muzhestva
15:23, 1, Politehnicheskaja
15:28, 1, Akademicheskaja
15:35, 1, Grazhdanskij prospekt
15:42, 1, Devjatkino
```

5.4. Code

The code is available at https://github.com/HaselLoyance/ga-spb-tsm

6. Bibliography

[Hussain et al., 2017] Hussain, A., Muhammad, Y., Nauman Sajid, M., Hussain, I., Mohamd Shoukry, A. and Gani, S., 2017. Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Computational Intelligence and Neuroscience*, 2017, pp.1-7.

[Larrañaga et al., 1999] Larrañaga, P., Kuijpers, C., Murga, R., Inza, I. and Dizdarevic, S., 1999. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review*, 13(2), pp.129-170.

[Miller and Goldberg, 1995] Miller, B. and Goldberg, D., 1995. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, 9(3), pp.193-212.

[Abdoun et al., 2012] Abdoun, O., Jaafar, A. and Chakir, T., 2012. Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem. *Int. J. Emerg. Sci.*, 2(1), pp.61-77.

[Jebari, 2013] Jebari, K., 2013. Selection Methods for Genetic Algorithms. *Int. J. Emerg. Sci.*, 3(4), pp.333-344.