Solving TSP Using Genetic Algorithm with Tournament Selection Variants

Krisztián Gajdár
Eszterházy Károly University
Eger, Hungary
krisztian.gajdar@gmail.com

***Abstract*—Genetic algorithms are evolutionary techniques used for optimization according to the survival of the fittest idea. These methods give a good approximation in a short amount of time. The genetic algorithms are useful for NP-hard problems, especially the traveling salesman problem.**
Tournament selection is a method of selecting an individual from a population of individuals in a genetic algorithm. Tournament selection (TS) involves running several ”tournaments” among a few individuals (or ”chromosomes”) chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover. During crossover we generate children from 2 tournament winners until we have a new population, we repeat this process until we have a good approximation.
In this article, we try out and evaluate the results of 3 variants of common GA selection method called tournament selection, one with no tournament selection, one with binary tournament selection, and 20 percent tournament selection on 5 differently sized TSP problems (5 nodes, 10 nodes, 20 nodes, 50 nodes, 100 nodes). We compare the results according to the solution closeness and the number of iterations required to get there and draw conclusions, which TS variant should be used in which case.

***Index Terms*—Genetic algorithm, NP-hard problems, traveling salesman problem, tournament selection**

I. Introduction

The Travelling Salesman Problem (TSP) [1] is a well known NP-hard optimization problem with the goal of finding the shortest route for a salesperson to take when visiting N number of cities. This type of problem appears in many forms with some engineering applications that include Vehicle routing [2], scheduling problems [3], integrated circuit designs [4].
A large number of methods have been developed for solving TSPs.
The Traveling Salesman Problem (TSP) is a classical combinatorial optimization problem, which is simple to state 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. This problem is known to be NP-hard, and cannot be solved exactly in polynomial time.

Although Evolutionary Algorithms [**?**] such as Genetic Algorithm [5] (GA) efficient to solve difficult optimization problems the Evolutionary Algorithms, such as the GA, are not efficient to find the optimal solution to TSPs compared to other methods, with certain improvements they can give a fairly close result in a short amount of time.

One crucial part of GA is the selection method, for this paper, I use different sizes of Tournament Selection (TS) and present the results. The exact methods will be discussed later on. In short: There are 5 TSP problems with each having 5, 10, 20, 50, and 100 nodes accordingly. On these datasets, I try out a GA implementation with No TS, Binary TS, and 20% TS which are also discussed later.

It is important to find a method to get good results because these methods can be used in fields where a form of TSP is a common problem. GAs are old, but still widely used and give good results, so testing it on this problem can deliver good results.
I want to prove that using a fast Binary TS can get us good results close to the results 20% TS.

II. Methods

A. Genetic Algorithm

Every organism has it's set of rules, which describes how that organism is built up. These rules are encoded in the genes of an organism, which are connected together and form a long string called chromosome. Each gene represents a specific trait of an organism, like hair and eye color, length of limbs, and so many other things.
When 2 organisms have a child, the child will share their genes, the child might have half the genes from one parent and the other half from the other parent. There are some cases when the gene mutates. Most often this doesn't result in any change for the individual, but sometimes it may generate a new trait.

Life on Earth evolved with a process called natural selection, gene sharing, and mutation. This process created our diverse world of plants and animals. Natural selection is also called ”survival of the fittest”, which means, the one species which adapted best to the circumstances around it, has a higher chance to survive and transfer its gene pool for the next generations. Gene mutation also has a big role in evolution, this helps species to adapt to the world around it.

At the beginning of a run of a GA, a population of random individuals (chromosomes) is created, All one of them represents a solution for the problem at hand.

The steps for the algorithm are the followings:

Step 1. Test each individual to see how good it is at solving the problem and assign a fitness score. The fitness score is a measure of how good that individual is at solving the problem.

Step 2. Select two individuals from the current population. The chance of being selected is proportional to chromosomes fitness. There are many ways to do this, Roulette wheel selection, and tournament selection are a common way to do this step.

Step 3. According to the crossover rate, the chromosome blocks are selected from each chosen chromosome starting from a randomly chosen point.

Step 4. Go through the chosen chromosomes parts and change them randomly according to the mutation rate.

Repeat step 2, 3, 4 until a new population with the same amount of members as the old one has been created.

B. Selection Methods

In a selection method, we select 2 parents to create an offspring for the next generation. Parent selection is very important for the convergence rate of the algorithm because by selecting good parents we get better solutions in each generation.
One thing to look for is to prevent one extremely fit individual from taking over the entire population, in a few generations, because by doing this we lose diversity. Maintaining good diversity is very important for a genetic algorithm's success. When one individual is taking over the entire population in a few steps is called premature convergence, and we want to avoid this in our GA.

Now let's look at some selection methods used in genetic algorithms:

Random selection is when we randomly select parents from the population. Fitter individuals have no better chance to be selected as the least fit ones. This method should don't generate a good result, so this one shouldn't be used.

Fitness Proportionate Selection is one of the most popular ways of parent selection. In this method, every individual can become a parent with a probability of its fitness. So, the fitter the individual, the higher the chance it will get selected, mate and share its features with the next generation.
With this selection strategy, we select the more fit individuals in the population, so we get better results over time.
One good thing to know is that fitness proportionate selection doesn't work for cases where fitness can be negative.
There are 2 implementations of fitness proportionate selection: Roulette Wheel Selection and Stochastic Universal Sampling. When using roulette wheel selection, a circular wheel is divided as described before. A fixed point is chosen on the wheel and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated. The fitter individual takes up a bigger space on the wheel, so it has a better chance of being selected.
Stochastic Universal Sampling is very similar to roulette wheel selection, but instead of having one fixed point, we have multiple of them, one for each parent, so we have to spin the wheel only once. This method encourages highly fit individuals to be chosen at least once.

Rank Selection can work with negative values and is mostly used when individuals have very close fitness values. In this case, fitter individuals have almost as much chance to be selected as the less fit ones. In this method, we don't select the individuals based just on their fitness. What we do is rank them according to their fitness, and we say that the higher an individual's rank is, the higher the chance it will be selected.

Tournament Selection is a very popular method and it is the method we use in our algorithm. It can also deal with negative fitness. In a tournament selection, we select a specific number of individuals from the population randomly and select the best of them to be the parent.

C. Implementation

For this paper I have 5 TSP problems (5, 10, 20, 50, 100 nodes), each node has an X and Y value representing its position, these are random values between 1 and 1 million, these values are stored in text files. Let's refer to these value pairs as Cities.

The GA is written in Python3 with spyder3 IDE without using any external libraries. I use a population size of 50 for a good balance of run-time and results. The mutation rate is set to 0.1. One run is constrained to 1000 iterations, but the program also ends if we get the same results for a consecutive 100 runs. For calculating the distance between cities we use the Euclidian distance algorithm. The fitness value in our case will be 1 divided by the distance, the shorter distance, the better fitness is.

First, we start by initializing our graph with values of the TSP problem at hand. Now we have to set our population, we do this by randomly adding each element of the graph to an array, we'll refer to this array as tour later on and we generate as many tours as our population number and we store these arrays in another array called tours. After we set our genetic algorithm with the parameters we start evolving in a loop, counting the amount of consequent same

results and stopping the program after it reaches 100 or the loop reaches 1000 cycles. By getting the fittest tour with the shortest distance we have our results for the iteration. After this, we evolve our population the get the next generation. We start by checking if we have elitism enabled, if yes, then we save the fittest tour to the next generation, then we go through the population selecting 2 parents by two tournament selections. We do a crossover with these 2 tours and we save the child for the next generation, we repeat this until we have a whole new generation. One thing we have to do is go through the new generation to mutate with the mutation rate set in the beginning.
Mutation happens by going through the population, and in each iteration we generate a random real number between 0 and 1, and if this random number is smaller than our mutation rate we do the mutation, which is swapping the iteration's current city with a random one in the tour, by doing this we get a different tour that we save.
We haven't discussed 2 things, how the crossover and the tournament selection happens.
The crossover's job is to make 1 child from the attributes of 2 parents. We do this by selecting a random interval in the first parent and add this sequence to the child to the same positions as it is the first parent, and for the empty space we fill it with second parent's values, this case we certainly we have values which can't be put in the same position as it was in the mother if we already put the value into the child from the first parent, in this case just put it into an empty space, we repeat this, until there are no cities left.
We select the parents by running a tournament selection. In the TS we generate a population with the size we set in the beginning, in our case this will be 1, for no TS, 2 for binary TS, 10 for 20% TS. We select individuals from the current generation randomly until we have as many as the tournament size. Then we select the individual with the highest fitness score (shortest tour distance), and we set this individual as the parent.
We ran the algorithm 15 times for each combination of TSP size and TS method for a total of 225 times, the complete experiment took over 40 minutes to run.

III. Results

In this section, I present the results of 15 runs of the genetic algorithm.

TABLE I
5 node TSP

	NT		**BT**		**TT**	
	Distance	***Runs***	***Distance***	***Runs***	***Distance***	***Runs***
Min	1774889	99	1774889	99	1774889	99
Max	1774889	110	1774889	258	1774889	126
Avg	1774889	101	1774889	101	1774889	105

NT - No Tournament. BT - Binary Tournament. TT - 20% Tournament.

First, in the 5 node TSP,, we can see that all TS variants found the best solution NT and BT having very close to the same results. But in the case of TT, the number of iterations grow by close to 14%.

TABLE II
10 node TSP

	NT		**BT**		**TT**	
	Distance	***Runs***	***Distance***	***Runs***	***Distance***	***Runs***
Min	2921770	115	2921770	102	2921770	101
Max	3052249	408	2921770	108	2921770	118
Avg	2944242	201	2921770	148	2921770	108

NT - No Tournament. BT - Binary Tournament. TT - 20% Tournament.

In the 10 node TSP, we can see that the BT and TT had no trouble finding the best distance in all cases, and in this case, TT was the fastest on average. The NT had a problem finding the best solution in all cases, producing a slightly bigger average than the other 2 methods.

TABLE III
20 node TSP

	NT		**BT**		**TT**	
	Distance	***Runs***	***Distance***	***Runs***	***Distance***	***Runs***
Min	5045914	116	4245094	156	3847702	142
Max	7291458	655	5934642	735	4922140	477
Avg	6069722	314	5002786	433	4245039	234

NT - No Tournament. BT - Binary Tournament. TT - 20% Tournament.

In the 20 node TSP, we can see some big differences. The TT wins both in distance and runtime. In this case, the NT and the BT didn't produce close to good results. BT outperforms the NT by 21% in terms of result but with the cost of a higher number of iterations.

TABLE IV
50 node TSP

	NT		**BT**		**TT**	
	Distance	***Runs***	***Distance***	***Runs***	***Distance***	***Runs***
Min	17426292	123	16292839	129	11561262	184
Max	20891878	661	19812342	489	14676735	934
Avg	19250051	271	18152364	267	13169675	489

NT - No Tournament. BT - Binary Tournament. TT - 20% Tournament.

In the 50 TSP, we can see that TT provides good results, while NT and BT lack behind. NT and BT provide fairly close results to each other's, with BT having a bit better results and runtime. Comparing runtimes BT comes out on the top but NT is very close too.

TABLE V
100 node TSP

	NT		**BT**		**TT**	
	Distance	***Runs***	***Distance***	***Runs***	***Distance***	***Runs***
Min	41623330	106	38808454	112	34143970	216
Max	45416106	481	44213767	346	37731234	691
Avg	43139795	252	41674045	232	36546875	355

NT - No Tournament. BT - Binary Tournament. TT - 20% Tournament.

In the 100 node TSP, we can see almost the same results as in the 50 node TSP but now the BT performed faster compared to others.

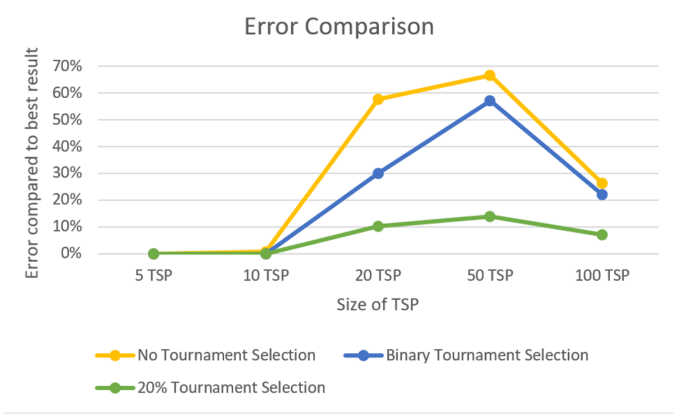

Fig. 1. Runtime compared

On the graph, we can see error rates compared. We calculated this by subtracting the average distance for the TS at hand from the best possible distance and dividing this number by the best possible distance. We call this value error.
In this, we can see that the three TS variants show no significant error in the 5 TSP and in the 10 TSP. The 20 TSP and 50 TSP show a big difference in terms of error, but the error comes closer at the 100 TSP.

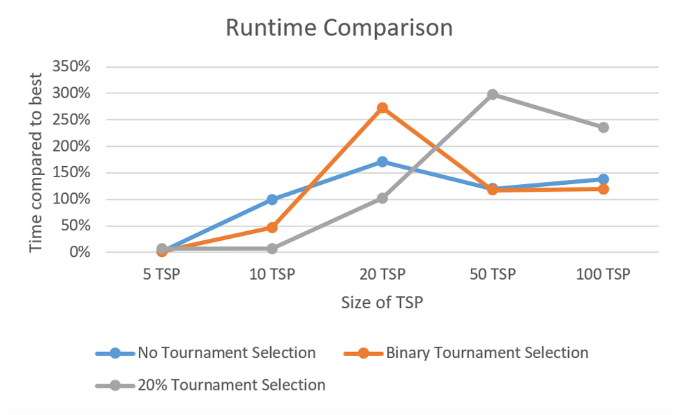

Fig. 2. Error rate compared

On the graph, we can see runtimes compared. We calculated these percentages by subtracting the average number of iterations for the TS at hand from the lowest iteration number in the current TSP and dividing this number by the lowest iteration number in the current TSP.
In this graph we can see that the runtimes for the three TS variants stay close at TSP 5, but grow apart from each other later on, Binary TS and No TS joining at 50 TSP and 100 TSP. We can also see that Binary TS is the highest at 20 TSP and 20% TSP is the highest at 50 TSP and 100 TSP.

IV. Discussion

What we have seen in the results is that by increasing the number of nodes in the Travelling Salesman Problem we get worse results with all of the three tournament selection variants.
The TT outperformed the others but with the cost of having slower runtime by 83% in the 50 node TSP and by 53% in the 100 nodes TSP than the binary tournament selection. But in the case of smaller TSPs, it performed faster, in the 10 node TSP by 37% and in the 20 node TSP by 85% than the binary tournament selection.
If we look at the BT in the 50 node TSP and 100 node TSP we can see that it performed 83% faster in the first case and 53% faster than the TT having 38% worse results in the first case and 14% worse results in the second case on average.
We can also see that by using no tournament selection we always get worse results and runtime is becoming worse.

From these result, we can draw the conclusion that using any tournament selection provides better results and also in a faster runtime than not using one. We can also say that using a 20% tournament selection we always get better results, but with the cost of having slower runtimes over 20 nodes than in the case of a binary tournament selection. We should always use 20% tournament selection except if we value 50% - 80% faster runtimes more than 10% - 40% better results. Most of the time this is not the case.

In the future for better results we need to increase the sample size from 15 to a higher number, and also increase the number of TSP problems to see if findings of this paper still stand after that.
We can also try out different selection methods like roulette selection and others to see if they get better results than our Tournament Selection Variants or we can try to increase the tournament selection even further.

V. Acknowledgement

I want to thank my teachers Zsolt Tóth, Győző Yang Zijian for letting this paper happen, and helping with the problems that came up during writing it.

References

[1] Laporte, G. "The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms", European Journal of Operational Research, 59, 231-247., 1992. http://dx.doi.org/10.1016/0377-2217(92)90138-Y
[2] G. Clarke and J.W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," Oper. Res., vol. 12, pp. 568–581, 1964.G. http://dx.doi.org/10.1287/opre.12.4.568
[3] D. Whitely, T. Starkweather, and F. D'Ann, "Scheduling problems and traveling salesman: The genetic edge recombination operator," in Proc.3rd Int. Conf. Genetic Algorithms, 1989, pp. 133–140.
[4] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, pp. 498–516, 1983. http://dx.doi.org/10.1126/science.220.4598.671
[5] A. E. Eiben and J. E. Smith. Introduction to evolutionary computing, volume 53. Springer, 2003.
[6] Mitchell, M. (1996). An Introduction to Genetic algorithms. MIT Press, Cambridge, MA.