Travelling Salesman Problem: A Comparative Study of Genetic Algorithm and Simulated Annealing Approaches

Buzdea Stefan, Pricop Tudor

December 28, 2023

Abstract

As the task of computing the optimal solution for the Travelling Salesman Problem (TSP) is very hard for many instances, we present an approach that tries to find good solutions by using a Genetic Algorithm (GA) improved with the 2-OPT heuristic. The 2-OPT heuristic greedily initializes the starting population, which is then improved by the more explorative GA. The proposed algorithm is tested on a set of 10 2-dimensional euclidean space instances (ordered by difficulty: berlin52, st70, kroA100, d198, kroB200, pr439, u2319, pcb3038, usa13509, d18512), obtaining good results for the smaller instances (over 98% accuracy) and approximately 84% accuracy for the hardest instance, maintaining balance between computing time and result accuracy. Afterwards, we compare the results of the GA with the outcomes of a modified Simulated Annealing algorithm to observe the relative effectiveness and performance variations between the two approaches.

1 Introduction

This paper aims to solve the classic combinatorial optimization problem of Symmetric Travelling Salesman, which can be described as: Given a finite set of n cities and a function which denotes the cost of travelling from one city to another, find the optimal tour (smallest cost) that visits each city exactly once and returns to the starting city. As TSP has very important real-world applications but it is a NP-hard problem, we approach this by using evolutionary and heuristic algorithms.

For doing this, Genetic Algorithm (GA) is a well-known algorithm, which is inspired from biological evolution process, mimicking the Darwinian theory of survival of fittest in nature. The basic elements of GA are chromosome representation (a permutation of nodes in this case), fitness selection (value assigned to a chromosome), and biological-inspired operators. The biological-inspired operators are selection, mutation, and crossover. In selection, the chromosomes are selected on the basis of its fitness value for further processing. In crossover operator, a random locus (position) is chosen and it changes the subsequences between chromosomes to create off-springs. In mutation, some nodes of the chromosome permutation will change positions on the basis of probability.

Over the course of this paper all of these notions are covered in more detail, starting from Section 2 (Methods) which gives a general overview of the algorithm, followed by explaining the permutation format of the chromosomes, the fitness function choice, the tournament selection choice and genetic operators (order crossover and displacement mutation). As the GA itself may not be as efficient in finding good solutions, a part of the initial population is generated using the 2-OPT heuristic, as explained in the Heuristics section. In section 3 we explore the results given by the algorithm on a set of instances, analyzing how it performs as the instances get harder.

2 Methods

2.1 Genetic algorithm overview

Genetic algorithms are loosely based on ideas from population genetics. First, a population of individuals is created randomly. In our case, each individual is a permutation of nodes and can be thought of as a candidate solution for some problem of interest (called chromosome).

Variations among individuals in the population result in some individuals being more fit than others (e.g., better problem solutions, determined by using a fitness function). These differences are used to bias the selection of a new set of candidate solutions at the next time step, referred to as selection. During selection, a new population is created by making copies of more successful individuals and deleting less successful ones. However, the copies are not exact. There is a probability of mutation (random permutation swaps), crossover (exchange of information between two individuals), or other changes to the permutation during the copy operation. By transforming the previous set of good individuals to a new one, the mutation and crossover operations generate a new set of individuals, or samples, that ideally have a better than average chance of also being good.

When this cycle of evaluation, selection, and genetic operations is iterated for many generations, the overall fitness of the population generally improves, and the individuals in the population represent improved "solutions" to whatever problem was posed in the fitness function.

2.2 Chromosome representation

As mentioned, for the chromosomes we use the permutation representation. In permutation encoding of TSP, the individual chromosome representing a feasible solution is encoded as a N dimensional integer vector $\pi_i = (\pi_{i1}, \pi_{i2}, ..., \pi_{in})$ (representing N consecutive city nodes in a complete valid tour). An initial population of high quality can accelerate the population evolutionary processing of GA to reach a satisfactory global optimum or suboptimal solutions.

The generating process of the initial population in GA is as follows: 80% of the initial population is randomly generated and the other 20% individuals are created by the heuristics method based on the greedy strategy (2-OPT). The generating method of the initial population not only maintains the population diversity but also speeds up convergence on the final solution by providing promising initial solutions.

In our experiments the population size is 100, so 20 individuals are generated using 2-OPT, which is further explained.

2.3 Fitness function

The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

Our goal is to minimize the cost of travelling, so the fitness score is inversely proportional to the function value determined by a chromosome. The following formula for calculating the fitness of a chromosome i (with function value y_i) is used, where $maxy/miny$ are the maximum/minimum function values in the current generation:

$$\text{fitness}_i = \frac{maxy - y_i}{maxy - miny + \epsilon} + 1$$

2.4 Selection

The idea of the selection phase is to select individuals and let them pass their genes to the next generation. Individuals with high fitness have more chance to be selected for reproduction.

In the genetic algorithm used for testing the benchmark functions the tournament selection is implemented, picking randomly k individuals and choosing some of the best ones among these to be part of the next generation of candidates. This is repeated until we reach the desired number of individuals in the population. The bigger k is, the higher the pressure of selection, because the fittest individual of the generation has a higher chance of being selected.

In our experiments, $k = 50$ individuals are randomly picked from the population of 100 individuals, from which the best 25 are selected.

2.5 Genetic operators

2.5.1 Order Crossover

The order crossover (OX1) exploits a property of the path representation, that the order of cities (not their positions) are important. It constructs an offspring by choosing a subtour of one parent and preserving the relative order of cities of the other parent. For example, consider the following two parent tours and suppose that we select a first cut point between the second and the third bit and a second one between the fifth and the sixth bit: (12|345|678) and (24|687|531). The offspring are created in the following way: first, the tour segments between the cut point are copied into the offspring, which gives $(\ast\ast|345|\ast\ast\ast)$ and $(\ast\ast|687|\ast\ast\ast)$. Next, starting from the second cut point of one parent, the rest of the cities are copied in the order in which they appear in the other parent, also starting from the second cut point and omitting the cities that are already present. When the end of the parent string is reached, we continue from its first position. In our example this gives the following children: (87|345|126) and (45|687|123).

The crossover probability in our experiments is 0.3.

2.5.2 Displacement And Inversion Mutation

The displacement mutation operator first selects a subtour at random. This subtour is removed from the tour, reversed, and inserted afterwards in a random place.

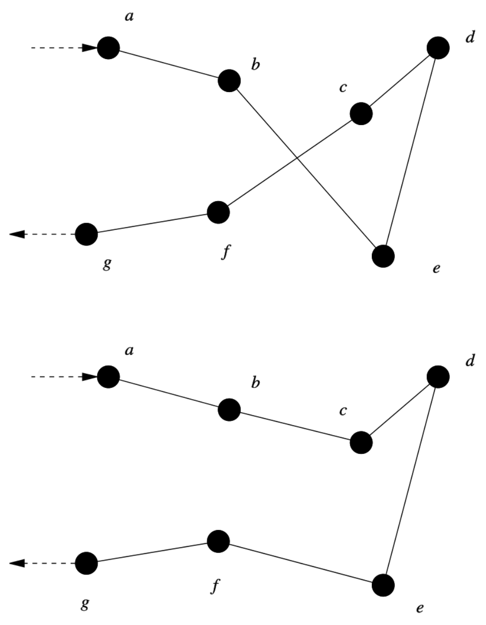
The mutation probability in our experiments is 0.1.

2.6 2-OPT Heuristic

This improvement approach is motivated by the following observation for Euclidean problems. If a Hamiltonian cycle crosses itself it can be easily shortened. Namely, erase two edges that cross and reconnect the resulting two paths by edges that do not cross (this is always possible). The new cycle is shorter than the old one.

A 2-opt move consists of eliminating two edges and reconnecting the two resulting paths in a different way to obtain a new cycle. The operation is depicted in Figure 1, where we obtain a better solution if edges be and fc are replaced by edges bc and fe. We repeat this move until no more improvement can be made anymore.

We use this heuristic to generate 20% of our initial population so that the first generation starts with some quality.



2.7 Simulated Annealing overview

We are implementing an iterative simulated annealing algorithm, complemented by the first improvement variant to efficiently identify the neighbor candidate of a chromosome. The candidates are represented as arrays of integers, signifying permutations for the order in which cities are visited.

To initiate the process, we randomly generate a chromosome and iteratively apply the 2-opt operation until no further improvements are achieved. This ensures a solid initial starting point for our algorithm. Subsequently, we generate neighboring solutions, selecting the one with the most significant improvement based on the chosen improvement method, and updating the best solution encountered thus far.

For generating neighboring solutions, we employ the displacement method introduced in the mutation section. The search for the "neighbor" candidate ceases as soon as we discover a chromosome with a fitness score superior to the current chromosome.

3 Experimental Results

3.1 Results for Simulated Annealing

The algorithm was implemented in C++, using a sample size of 30 for each instance. The following parameters were used:

- `iterations` = 5000
- `temperature` = 100
- `cooling coefficient` = 0.99

Instance	**Cities**	**Best Result**	**Avg Result**	**Optimum**	**Accuracy (%)**	**Std Dev**
berlin52	52	8432	8980.56	7542	80.93	124.67
st70	70	790	852.98	675	75.11	7.22
kroA100	100	25645	26520.34	21282	76.33	94.67
d198	198	18941	19608.27	15780	75.74	30.72
kroB200	200	35616	37935.74	29437	71.13	132.22
pr439	439	131543	139793.02	107217	69.62	201.58
u2319	2319	327340	335003.61	234256	56.99	121.49
pcb3038	3038	197369	202974.87	137694	52.59	180.56
usa13509	13509	28021521	29092483.29	19982889	54.41	651.6
d18512	18512	935288	951532.91	645488	50.02	154.75

Table 1: Simulated Annealing Results

3.2 Results for Genetic Algorithm

The algorithm was implemented in C++, using a sample size of 30 for each instance. The following parameters were used:

- `maxGenerations` = 2000 generations
- `popSize` = 100 individuals in one generation
- `improvedPopSize` = 20 individuals generated using the greedy strategy
- `crossoverProb` = 0.3
- `mutationProb` = 0.1
- `accuracy` = $(1 - \frac{\text{avg-optimal}}{\text{optimal}}) \times 100$
- `tournamentPressure` = 50 individuals picked for tournament selection, from which 25 are selected

Instance	**Cities**	**Best Result**	**Avg Result**	**Optimum**	**Accuracy (%)**	**Std Dev**
berlin52	52	7606	7680.53	7542	98.16	86.30
st70	70	684	693.00	675	97.33	5.33
kroA100	100	21787	21989.76	21282	96.67	181.04
d198	198	16191	16353.33	15780	96.37	93.10
kroB200	200	29871	31037.93	29437	94.56	372.69
pr439	439	113339	115785.23	107217	92.01	1360.00
u2319	2319	256981	258351.63	234256	89.71	631.75
pcb3038	3038	155431	156514.40	137694	86.33	567.12
usa13509	13509	22392422	22512003.66	19982889	87.34	60298.05
d18512	18512	744001	745787.14	645488	84.46	1009.67

Table 2: Genetic Algorithm Results

The proposed algorithm performs very well on smaller instances, the accuracy dropping significantly as the number of cities increase, but still providing decent approximations. However, it still outperforms a simple GA most of the times, being a more preferred approach. More than that, compared to other approaches, the execution times are very low, taking into consideration the fact that one single 2-opt operation is done in $O(n_{\text{cities}}^2)$ and repeated until no improvement is made anymore.

4 Comparison

To evaluate the efficacy of the Genetic Algorithm (GA) in addressing the Travelling Salesman Problem (TSP), we conducted a comparative analysis with results obtained from a modified Simulated Annealing algorithm (SA). Both algorithms were applied to the same set of instances, leading to the following observations:

- **Accuracy:** The GA consistently demonstrated high accuracy, surpassing 98% for smaller instances. However, as the complexity of the instances increased, the accuracy declined, reaching approximately 84% for the most challenging case. In contrast, the SA algorithm exhibited a notable decrease in accuracy when compared to the genetic algorithm, indicating its imprecision on larger instances. It is evident that SA is more affected by the dimension of an instance than the GA, showcasing its imprecision even in smaller instances.
- **Algorithm Behavior:** The GA, as an evolutionary algorithm, demonstrated a balanced approach between exploration and exploitation, making it well-suited for TSP instances. On the other hand, the SA algorithm utilized a modified version with the first improvement strategy. Due to a larger search space, the SA performed noticeably worse in higher dimensions. It was observed that the correspondence between accuracy and instance dimensions is not linear, with a significant drop occurring when transitioning to higher dimensions.
- **Solution Quality:** While both algorithms provided decent approximations for small instances, the GA consistently outperformed the SA algorithm in terms of solution quality. The GA's effectiveness in exploring the solution space, combined with the 2-opt heuristic, contributed to obtaining more accurate results. In contrast, the SA algorithm, even with the optimization of the 2-opt during the initial instance generation, was significantly inferior in terms of solution quality.

5 Conclusions

In conclusion, the Genetic Algorithm (GA) showcased commendable accuracy, particularly for smaller instances, while facing a decline in performance with increased complexity. Simulated Annealing (SA), despite an initial focus on optimization, demonstrated noticeable inaccuracies in larger instances, revealing its sensitivity to problem dimensions. The GA's balanced exploration-exploitation strategy, coupled with the 2-opt heuristic, consistently yielded superior solutions compared to SA.

This suggests that, for solving the Travelling Salesman Problem, the GA is a more robust and reliable algorithm across a spectrum of instance sizes.

References

[1] 2-opt. en. Page Version ID: 1128223564. Dec. 2022. https://en.wikipedia.org/w/index.php?title=2-opt&oldid=1128223564.

[2] Convert CSV to LaTeX Table. en. https://tableconvert.com/csv-to-latex.

[3] Eugen Croitoru: Teaching: Genetic Algorithms. https://profs.info.uaic.ro/~eugennc/teaching/ga/.

[4] Michael Junger, Reinelt Gerhard, and Rinald Giovanni. “The traveling salesman problem”. In: Handbooks in operations research and management science 7 (1995): 225-330 (). http://www.iasi.cnr.it/ResearchReports/R00375.pdf.

[5] P. Larranaga et al. “Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators”. In: Artificial Intelligence Review 13.2 (Apr. 1999), pp. 129–170. issn: 1573-7462. doi: 10.1023/A:1006529012972. https://doi.org/10.1023/A:1006529012972.

[6] Bao Lin, Xiaoyan Sun, and Sana Salous. “Solving Travelling Salesman Problem with an Improved Hybrid Genetic Algorithm”. In: Journal of Computer and Communications 4.15 (Nov. 2016), pp. 98–106. doi: 10.4236/jcc.2016.415009. http://www.scirp.org/Journal/Paperabs.aspx?paperid=72322.

[7] “The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem”. In: Mathematical and Computer Modelling 31.10-12 (May 2000), pp. 197–203. issn: 0895-7177. doi: 10.1016/S0895-7177(00)00088-1. https://www.sciencedirect.com/science/article/pii/S0895717700000881.