Genetic Algorithm and Simulated Annealing Algorithm in the context of The Travelling Salesman Problem

Aciocanesa Bianca-Ioana

January 10, 2024

Abstract

The Travelling Salesman Problem, a classic **NP-hard** problem, has been a focal point in algorithmic research. This study focuses on presenting a comparative analysis for solving this problem with two different algorithms: a Simulated Annealing algorithm and a Genetic algorithm. Both of them have proven to be effective in finding an approximate solution for 10 different instances of the problem. For example, for smaller instances such as eil51, with only 51 cities, both of the algorithms found similar results, that are pretty close to the optimal tour: Simulated Annealing obtained 433.69 and the Genetic Algorithm 433.523 (versus 426 - the optimal result). A difference was made between the results by the two algorithms for more complex instances of the problem, where, although Simulated Annealing performed faster, the Genetic Algorithm offered more precised results (for example, for the nrw1379 instance, the results were: Simulated Annealing - 64681.7, Genetic Algorithm - 63979.4 versus the optimal result - 56638).

1 Introduction

The Traveling Salesman Problem (TSP) stands as a challenge in the field of combinatorial optimization, requiring the determination of the most efficient route for a salesman to visit a set of cities exactly once and return to the starting point. As TSP's complexity grows with the number of cities, metaheuristic algorithms have become indispensable tools for approximating optimal solutions within a reasonable timeframe.

Among these metaheuristic approaches, **Simulated Annealing** and **Genetic Algorithm** have proven to be very effective in navigating the vast solution space of TSP. Simulated Annealing, inspired by the annealing process in metallurgy, and Genetic Algorithm, mimicking the principles of natural selection, offer distinct strategies for exploring and exploiting solution landscapes.

This study focuses on comparing those two algorithms designed to find an approximate solution for 10 different instances chosen from the TSPLIP , starting with a lower number of cities and progressively incresing the complexity with a lager inputs. The following 10 instances have been chosen for this experiment: eil51, st70, pr107, tsp225, rd400, pr439, rat783, nrw1379, u2152, pr2392.

At the beginning, when the small instances were run with the two algorithms, a significant difference in performance was not noticed between them. For instances with a larger number of cities, a difference in the operation of the chosen algorithms was observed, as will be explained in the following sections. It is worth noting that both algorithms produced satisfactory results, starting from eil51 with only 51 cities (where Simulated Annealing obtained 433.69 and the Genetic Algorithm 433.523, versus 426 - the optimal result) up to the biggest instance that was chosen (for example, for the nrw1379 instance, the results were: Simulated Annealing - 64681.7, Genetic Algorithm - 63979.4 versus the optimal result - 56638).

2 Methods

2.1 Simulated Annealing algorithm

2.1.1 Description

Simulated Annealing, inspired by the annealing process in metallurgy, is a powerful optimization algorithm applied to combinatorial problems like the TSP. In TSP, it starts with an initial tour and iteratively explores neighboring solutions by applying local search operations. The algorithm evaluates the quality of these solutions based on the total distance of the tour. Notably, Simulated Annealing introduces a temperature parameter, controlling the probability of accepting worse solutions. This temperature starts high and gradually decreases, allowing the algorithm to initially explore a broader solution space and later focus on refining the tour. Simulated Annealing's ability to probabilistically accept worse solutions facilitates escaping local optima, making it a versatile tool for finding near-optimal solutions in TSP and other optimization challenges.

2.1.2 Pseudocode

1. *intialize the temperature* T
2. *contor* $:= 1$
3. *generate the starting candidate* $c(contor)$
4. *evaluate* $c(contor)$
5. *while halting criterion*

- *while termination condition*
  - *select neighbour* $n(contor)$ *from* $c(contor - 1)$
  - *if* $eval(n)$ *is better than* $eval(c)$ *then* $c = n$
    else if $random[0, 1) < exp(-|eval(n) - eval(c)|/T$
    then $vc = best\_improvement(n)$
- *descrease* T
- $contor := contor + 1$

The algorithm starts with a "temperature" (a parameter T) and with every iteration it "cools down", trying to lower the chances of choosing a worse solution. Initially, the temperature is 10^5 ,which is multiplied at every step by a value smaller than 1 (in this case, *descrease* T is $0.999^{contor} * 100$ where contor is a parameter that starts with the value 1 and increases with 1 at every step of the algorithm) until it reaches an unsignificant value (a value lower than $10e^{-12}$) - this is the halting criterion.

A neighbour of the candidate solution is generated by choosing two random points and reversing the values from the sequence between those two positions.

The *best_improvement* function means that, when accepting worse solutions too, we still go in the next round of the algorithm with a better solution by swapping two by two cities from the permutation until a route with a smaller cost is found.

2.2 Genetic Algorithm

2.2.1 Representation

The algorithm starts from a given number of candidate solutions randomly generated as permutations - this is the **starting population** (each number from the permutation representing a city that has to be visited). Each chromosome from the population is then **evaluated**, meaning that it is calculated the lenght of the route if the cities are visited in the order of that permutation, and it is considered "better" if it returns a smaller value (since the purpose of the algorithm is to find the shortest route). A new **generation** of a population is obtained by applying the genetic operators over the previous population, in order to find progressively better solutions as generations advance until reaching the maximum number of generations set.

For this algorithm, the following set of parameters has been chosen:

- population size = 200
- number of generations = 2000

2.2.2 Pseudocode

1. $t := 0$
2. *generate the starting population* $P(t)$
3. *evaluate* $P(t)$
4. *while not stopping condition*
   - $t := t + 1$
   - *select* $P(t)$ *from* $P(t-1)$
   - *mutate* $P(t)$
   - *crossover* $P(t)$
   - *evaluate* $P(t)$

2.2.3 Genetic operators

1. **INVERSION MUTATION**
   Mutation serves as an **exploration mechanism**, allowing the algorithm to explore neighboring regions of the solution space. This randomness helps the genetic algorithm avoid getting stuck in local optima and facilitates the discovery of novel, potentially better solutions. For each individual from a given population, mutation is applied probabilistically in the following way: a random number from 0 to 1 is generated and if it is smaller than the chosen **mutation probability/rate**, that individual suffers mutation : **two random positions are chosen and the values from the sequence between those two positions are reversed**.

2. **ORDER CROSSOVER**
   Crossover, also called recombination, is a genetic operator used to combine the genetic information of two parents to generate new offspring and it serves as an **exploitation mechanism**. It is one way to stochastically generate new solutions from an existing population. We traverse all individuals in a generation, and with a predefined **probability of crossover**, we select them. Subsequently, two consecutive individuals from this selection undergo the crossover process as follows: the process begins by randomly selecting two crossover points; the segment between these points in one parent is directly inherited by the offspring; the order of elements outside the inherited segment is then determined by the order of corresponding elements in the other parent; this ensures a mix of genetic material from both parents, preserving the relative order of elements in each parent.

3. **SELECTION**
   The purpose of selection is to choose the most adapted individuals for survival in the population, with the hope that their descendants will have

higher fitness. In combination with variations of crossover and mutation operators, selection must maintain a balance between exploration and exploitation. High selection pressure leads to the creation of low diversity in the population composed of well-adapted but suboptimal individuals, resulting in a limitation of changes and, consequently, progress. On the other hand, low selection pressure slows down evolution. The algorithm used for this experiment contains the roulette wheel as selection method, that works in the following way:

- Every individual from a generation is evaluated (the bitstring is converted to the decimal representation and applied to the given function) : $eval[i]$
- The fitness function is calculated for every individual from a generation using the following formula:
  $fitness[i] = ((max - eval[i])/(max - min + 0.0001) + 1)^{10}$
  where max is the maximum evaluation from that population and min is the minimum evaluation
- Then, the fitness proportion of each individual is calculated by dividing its fitness by the sum of the fitness values of all individuals in the population - this creates a probability distribution where higher fitness individuals have larger proportions
- The roulette wheel is created - it is divided into segments, each corresponding to an individual in the population. The size of each segment is proportional to the individual's fitness proportion : the better the solution, the bigger the size of the segment, in order to create a larger probability for that element to be selected
- A "random spin" of the roulette is performed: a random number is generated and the individual on the segment where the probability lands is selected for the next generation

2.2.4 Optimizations

1. **ELITISM**
   Elitism in a genetic algorithm is a strategy that involves preserving a certain percentage of the best individuals (chromosomes or solutions) from one generation to the next without applying genetic operators like crossover and mutation to them. This ensures that the best-performing individuals in the population are directly carried over to the next generation, preventing the loss of their beneficial traits.

   In this regard, a reversed version of elitism has been implemented too, where, similar to elitism, where the top $x\%$ solutions directly advance to the next generation, the bottom $x\%$ solutions have a 50% chance of proceeding to the next generation. Otherwise, they are replaced in the new

generation by other randomly generated permutation. For this experiment, the elitism was 20, meaning that out of 200 individuals, only the best 20 of them were directly sent to the next generation without suffering crossover and mutation, and the worst 20 of them had 50% chance of survival.

2. **GREEDY APPROACH**
   The search space of the problem is very large; hence, the genetic algorithm begins its quest for the optimal solution from very high results, making it challenging to converge towards the true optimum. Therefore, in the implementation, the initial generation is populated with a few greedily chosen values (only 4 elements out of the 200) as follows. Starting from a randomly generated city, the algorithm selects the next city to visit based on the nearest unvisited city. After each selection, the algorithm updates the current city and marks the chosen city as visited. This process repeats until all cities have been visited, and the tour is completed by returning to the starting city.

   Introducing this approach into the genetic algorithm ensures that it does not consume generations by starting from randomly generated very high values; instead, it is guided by greedy values to converge towards an optimal solution more quickly, being influenced by the genetic material of a few good chromosomes.

3. **HYBRIDIZATION WITH SIMULATED ANNEALING**
   To enhance the genetic algorithm's ability to achieve optimal results, it was hybridized with the previously presented simulated annealing algorithm. In this context, for the best result obtained in the last generation, the algorithm was applied to see if the found solution could be further optimized (generating a neighbor of a candidate solution was indeed the mutation used by the genetic algorithm).

3 Experimental results

The following will present the results obtained by the two algorithms, compared to each other, for the ten instances in the table below:

NAME	NUMBER OF CITIES	LENGHT OF OPTIMAL SOLUTION
eil51	51	426
st70	70	675
pr107	107	44303
tsp225	225	3916
rd400	400	15281
pr439	439	107217
rat783	783	8806
nrw1379	1379	56638
u2152	2152	64253
pr2392	2392	378032

Table 3.0: The chosen 10 instances from TSPLIB

	best result	average	worst result
Optimum	426	426	426
Simulated Annealing	433.69	436.1845	447.466
Genetic Algorithm	433.523	438.2203	446.148

Table 3.1: Results for eil51 instance

	best result	average	worst result
Optimum	675	675	675
Simulated Annealing	688.74	705.838	744.446
Genetic Algorithm	677.11	699.574	715.283

Table 3.1: Results for st70 instance

	best result	average	worst result
Optimum	44303	44303	44303
Simulated Annealing	45489.4	46935.7	47608.9
Genetic Algorithm	44324.8	44843.4	45041.1

Table 3.2: Results for pr107 instance

	best result	average	worst result
Optimum	3916	3916	3916
Simulated Annealing	4019.11	4183.32	4301.26
Genetic Algorithm	4091.44	4165.46	4196.3

Table 3.3: Results for tsp225 instance

	best result	average	worst result
Optimum	15281	15281	15281
Simulated Annealing	16563.2	16831.5	17167.5
Genetic Algorithm	16198.4	16395.1	16599.9

Table 3.4: Results for rd400 instance

	best result	average	worst result
Optimum	107217	107217	107217
Simulated Annealing	118877	120051	124679
Genetic Algorithm	113074	114568	115651

Table 3.5: Results for pr439 instance

	best result	average	worst result
Optimum	8806	8806	8806
Simulated Annealing	9528.6	9928.83	10036
Genetic Algorithm	94338.1	96435	9844.4

Table 3.1: Results for rat783 instance

	best result	average	worst result
Optimum	56638	56638	56638
Simulated Annealing	64681.7	65253.175	66046.1
Genetic Algorithm	63979.4	65116	66115.9

Table 3.6: Results for nrw1379 instance

	best result	average	worst result
Optimum	64253	64253	64253
Simulated Annealing	74552	76123.5	78229
Genetic Algorithm	72314	74226.4	76494.8

Table 3.7: Results for u2152 instance

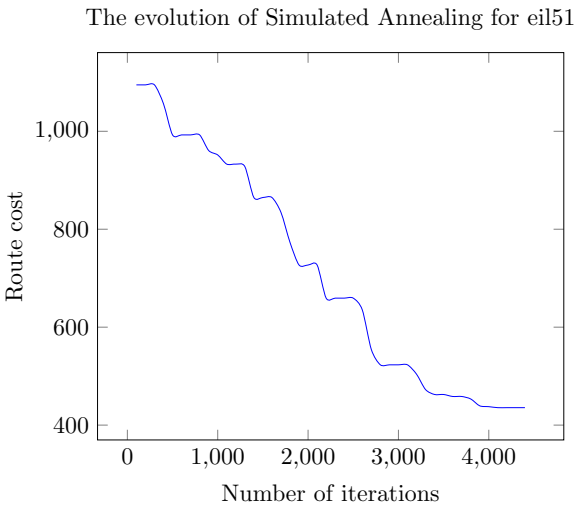
	best result	average	worst result
Optimum	378032	378032	378032
Simulated Annealing	514688	520652	525460
Genetic Algorithm	418743	424638	425171

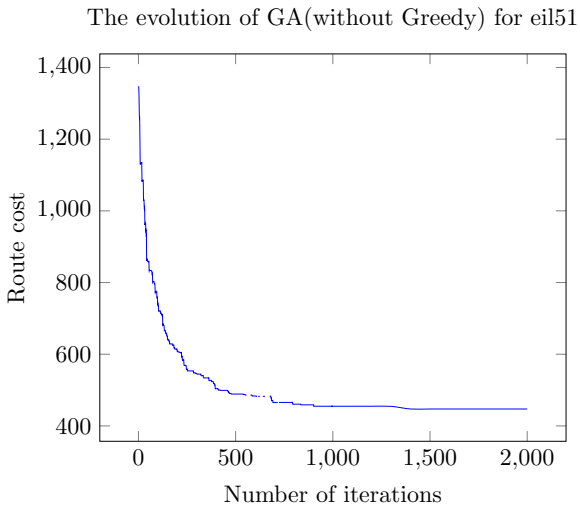
Table 3.8: Results for pr2392 instance

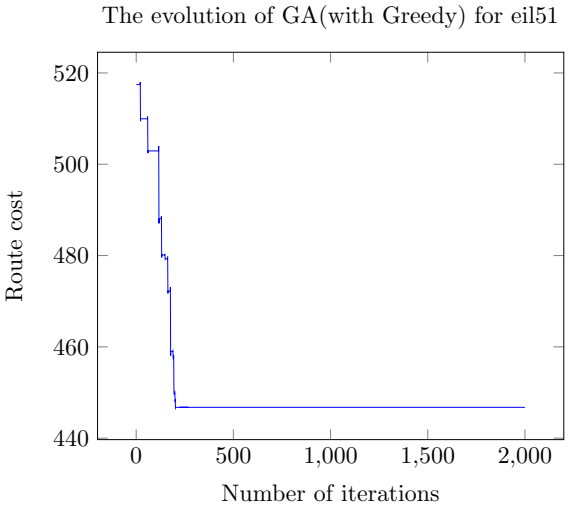
4 Comparison and analysis of the results - Genetic Algorithm vs Simulated Annealing

As can be observed from the obtained results, there is not a considerable difference in terms of accuracy between the results obtained with simulated annealing compared to the genetic algorithm. However, this fact is also attributed to the optimizations used. The difference between three variants can be observed from the graphs below: Simulated Annealing alone, the genetic algorithm without a greedily chosen initial population to a certain percentage, and the genetic algorithm with greedy population. Although the final result is almost the same, the difference can be observed from the starting point of the algorithm: the first two algorithms start with randomly generated initial solutions, thus with much higher starting values compared to when the algorithm is assisted by the greedy approach.

This disadvantage affects the first two algorithms because they spend more time reducing from a very high initial value, especially in instances with more nodes than the instance presented in the graphic. Therefore, the third algorithm has a more confined search area; it doesn't start the search from values that are very far from the real minimum being sought.







5 Conclusions

In conclusion, the study provides valuable insights into the performance of Simulated Annealing and Genetic Algorithm in tackling the Traveling Salesman Problem (TSP). The following key conclusions can be drawn from the experiments conducted on the selected instances:

- **Effectiveness for Small Instances**: For smaller instances, such as eil51, both Simulated Annealing and Genetic Algorithm demonstrated comparable effectiveness, producing results close to the optimal tour. However, it is important to note that the difference in performance between the two algorithms was not significant for these simpler cases.
- **Performance Discrepancy with Increasing Complexity**: As the complexity of instances increased, a noticeable divergence in performance emerged. Simulated Annealing exhibited faster execution, but the Genetic Algorithm demonstrated superior precision in finding more accurate solutions. This discrepancy becomes more apparent in instances with a larger number of cities, highlighting the strengths and weaknesses of each algorithm.
- **Hybridization Impact**: The hybridization of the Genetic Algorithm with Simulated Annealing proved beneficial in enhancing the overall accuracy of results. This hybrid approach contributed to refining solutions, especially in instances with a challenging solution space.
- **Trade-off Between Speed and Accuracy**: The choice between Simulated Annealing and Genetic Algorithm may depend on the specific requirements of the problem at hand. Simulated Annealing, with its faster execution, might be preferable for situations where quick solutions are prioritized, while the Genetic Algorithm, with its focus on precision, becomes more valuable when accuracy is paramount.

References

[1] Course's site
`https://profs.info.uaic.ro/~eugennc/teaching/ga/`

[2] Parameters used for the Simulated Annealing algoritm
`https://rbanchs.com/documents/THFEL_PR15.pdf`

[3] De Jong 1 function informations
`http://www.geatbx.com/docu/fcnindex-01.html`

[4] Schwefel's function informations
`https://www.sfu.ca/~ssurjano/schwef.html`

[5] Rastrigin's function informations
`https://www.sfu.ca/~ssurjano/rastr.html`

[6] Michalewicz's function informations
`https://www.sfu.ca/~ssurjano/michal.html`

[7] Informations about genetic algorithms
`https://en.wikipedia.org/wiki/Genetic_algorithm`

[8] Crossover image and informations
https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

[9] http://datajobstest.com/data-science-repo/
Genetic-Algorithm-Guide-[Tom-Mathew].pdf