Assignment 4. Solving Complex Problems

María Segura Bolaños

Computational Complexity
School of Engineering
University of Cádiz

maria.segurabolanos@alum.uca.es

14th January 2024

Summary

In this Computational Complexity assigment, we are going to solve the Traveling Salesman Problem (TSP), which is a classic optimization problem that involves determining the most efficient route that a salesman can take to visit a set of cities and return to the starting city, covering each city exactly once. With the number of cities, it is known that the possible solutions grow exponentially, making this combinatorial optimization problem belong to the NP-hard class.

In our approach, we are going to use the method known as simulated annealing (SA), a heuristic optimization technique that allows us to iteratively explore the solution space globally, escaping possible local optima. In general, an optimal solution for the TSP is not achieved, but it allow us to obtain a fairly good approximate solution.

Contents

Tables

Figures

1 TSPLIB Format

To solve the Traveling Salesman Problem, I have utilized the TSPLIB format for its standardized representation. Therefore, in the TSPLIB folder, there is a series of files organized in pairs, where .tsp files represent the initial instances of the problem with the coordinates of the cities, and .opt.tsp files contain optimal solutions for the corresponding maps. The initial instances follow a structured format, including information such as the name, comments, type, dimension, edge weight type, and node coordinate section, whereas the optimal tours include information such as the name, type, dimension, and tour section.

Within this program, I have implemented two key functions: tsp_to_graph and solution_to_tsp. The first is responsible for reading the initial instance file and constructing a complete undirected graph from it. In this graph, each node represents a city, and each edge represents the Euclidean distance between the two connected cities. The second function comes into play at the end of the program, allowing us to translate the best solution found into the TSPLIB format and saving the best tour founded in the folder BestSolutionsTSP.

This systematic approach to handling the TSP instances has simplified the development process, facilitating the reading of initial instances, the creation of a representative graph, and the translation of final solutions into a standardized format. Therefore, it enables us to execute the implemented program using the command ./tsp.exe followed by the name of the map we want to solve.

2 Initial solution

For the initial solution of the TSP, I chose to develop a greedy algorithm due to they are efficient at generating solutions quickly by making locally optimal decisions at each step. However, I later considered that there might be better alternatives because the greedy solution guarantees the best local result by visiting each city once, but the TSP requires returning to the origin, which makes a solution locally excellent but not necessarily globally optimal. Although simulated annealing can often escape local optima, its effectiveness can be compromised by the already high quality of the solution provided by the greedy algorithm, especially considering the last trip of the path. For this reason, I opted to start with a completely random tour. This approach offered a more diverse starting point for simulated annealing to explore, facilitating a more effective traversal of the solution space and potentially converging towards a more globally optimal solution for the Traveling Salesman Problem.

3 Simulate Annealing (SA)

Simulated Annealing (SA) is a global optimization algorithm inspired by the physics of solids and the metallurgical annealing process. Its essence lies in its ability to globally explore solution spaces and escape local optima through the probabilistic acceptance of suboptimal solutions. It begins with an elevated temperature that gradually decreases during algorithm iterations, simulating the cooling of a physical system. In each iteration, SA generates neighboring solutions and evaluates whether to accept or reject them based on the difference in quality between the current solution and the neighbor, as well as the current temperature. This strategy enables SA to extensively explore solution spaces initially and converge towards optimal solutions as the temperature decreases.

The choice of temperature is crucial for its effectiveness. In this case, I initialized the initial temperature with a value fifty times the number of cities in the TSP problem, aiming to strike a balance between the initial exploration of solution spaces and subsequent exploitation of local solutions. I experimented with higher temperatures, but they did not significantly improve solutions while noticeably increasing program execution time. Conversely, lower temperatures resulted in a more notable difference between the optimal solution and the algorithm's output.

Neighboring solutions are generated using the swapPath function, performing random exchanges between two cities or reversing the path segment between them based on a random number. This strategy seeks to diversify solutions, as swapping and reversing are operators that significantly alter the current path. The introduction of a random component in generating neighboring solutions aims to balance runtime and final solution quality. Swapping is much faster but yields inferior results compared to reversing a path segment.

Upon generating a neighboring solution, if its fitness is better, it is accepted outright. Otherwise, a random number between 0 and 1 is generated, and if this number is less than e raised to the power of the difference between the fitness of the current solution and the new solution divided by the current temperature, the worse solution is accepted. This approach aims to prevent getting stuck in local minima.

The cooling scheme is implemented by multiplying the current temperature by 0.99999 in each iteration. This gradual descent contributes to algorithm convergence, reducing the probability of accepting worse solutions as the execution progresses. The choice of this cooling factor, like the initial temperature selection, seeks a balance between solution quality and runtime.

4 Algorithm evaluation

In order to evaluate the algorithm, at the end of the program, there are a series of instructions that display on-screen the running time, the absolute error, and the relative error. In the following tables, I present the results based on the number of cities. As the algorithm has some random components, I have repeated the experiment three times for each tsp instance.

	eil51	st70	kroA100	ch130	ch150	tsp225	a280	pcb442	pr1002	pr2392
E1	23.570	24.573	26.870	26.518	27.434	27.607	28.367	30.270	32.844	35.269
E2	23.603	24.519	26.843	26.758	26.867	27.650	28.527	30.289	32.952	35.259
E3	24.848	24.566	27.984	26.715	27.438	27.647	28.486	30.335	32.928	35.216
RT	23.674	24.553	27.232	26.664	27.246	27.635	28.460	30.298	32.908	35.248

Table 1: Running time (in seconds)

	eil51	st70	kroA100	ch130	ch150	tsp225	a280	pcb442	pr1002	pr2392
E1	15.49	6.84	770.15	167.23	629.56	374.25	348.78	9616.15	182842.00	1.00e+06
E2	12.87	5.04	507.52	226.26	524.65	234.33	384.37	9198.89	184863.00	1.00e+06
E3	11.25	16.06	647.74	307.29	353.84	344.34	409.30	9655.63	172647.00	1.01e+06
AE	13.54	9.98	641.13	233.93	502.02	317.31	380.82	9489.56	180117.00	1.00e+06

Table 2: Absolute error

	eil51	st70	kroA100	ch130	ch150	tsp225	a280	pcb442	pr1002	pr2392
E1	0.036	0.010	0.036	0.027	0.096	0.097	0.135	0.189	0.706	2.655
E2	0.030	0.007	0.024	0.037	0.080	0.061	0.149	0.181	0.714	2.647
E3	0.026	0.024	0.030	0.050	0.054	0.089	0.158	0.190	0.666	2.685
RE	0.031	0.014	0.030	0.038	0.077	0.082	0.147	0.187	0.695	2.663

Table 3: Relative error

With the data obtained from the various experiments, I have calculated the average of the values for each instance of the TSP and presented those results in the following graphs:

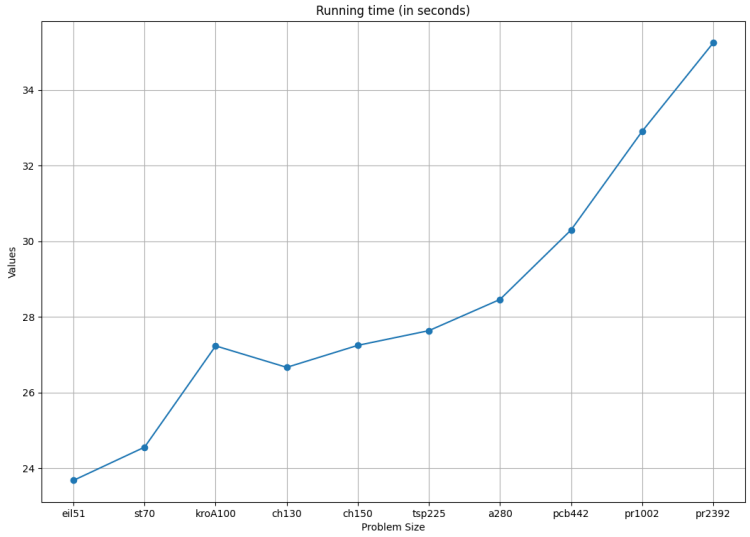

Figure 1: Running Time (in seconds)

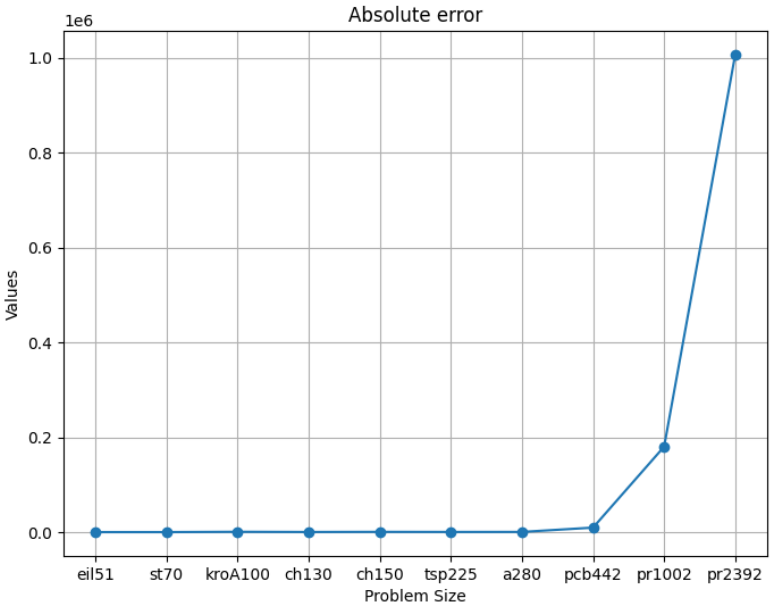

Figure 2: Absolute Error

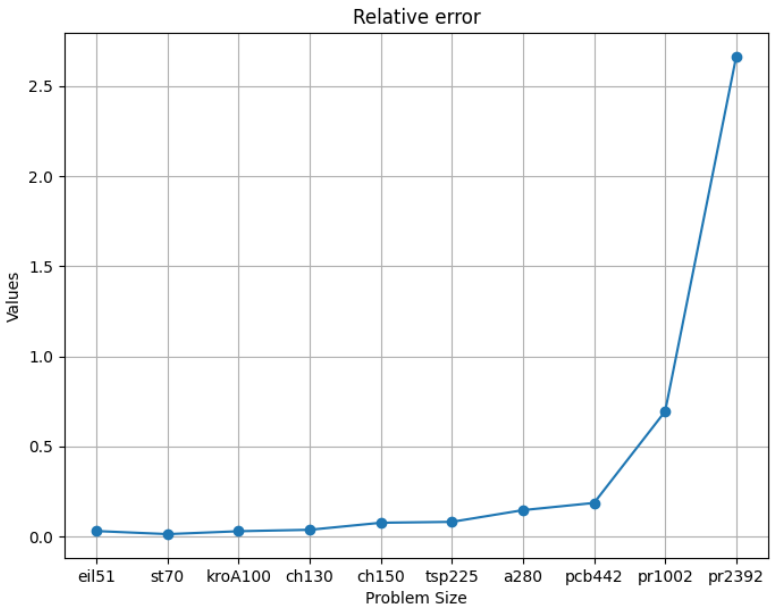

Figure 3: Relative Error

Examining the execution times, it is observed that they increase logically with the size of the problem. However, the growth is not exaggerated. For the smallest instance (51 cities), the execution time is approximately 24 seconds, while for the largest instance (2392 cities), the time is around 35 seconds. It is noteworthy that, despite a difference of 2341 cities between both instances, the increase in execution time is relatively small, being approximately 10 seconds.

In the context of absolute error, the difference between problems of different sizes is indeed more noticeable. Nevertheless, this is quite logical since larger problems have solutions with greater distances, leading to a greater variation between the optimal solution and the rest of the possible solutions.

Finally, it should be noted that the relative errors obtained are quite good. For problems of not excessively large sizes (up to 442 cities), the relative error does not exceed 0.2. However, in the two significantly larger problems (with more than 1000 cities), the relative errors are larger, reaching results approximately 2.663 worse than the optimal solution.

Bibliography

[1] Sebastián Gerard Aguilar Kleimann. *Algoritmo de recocido simulado o Simulated Annealing.*
URL: https://medium.com/estudio-de-datos/algoritmo-de-recocido-simulado-o-simulated-annealing-234f567677d9.

[2] Marc Kuo. *Algorithms for the Travelling Salesman Problem.*
URL: https://www.routific.com/blog/travelling-salesman-problem.

[3] Beatriz Pérez de Vargas Moreno. *Resolución del Problema del Viajante de Comercio (TSP) y su variante con Ventanas de Tiempo (TSPTW) usando métodos heurísticos de búsqueda local.*
URL: https://core.ac.uk/reader/211096990.