The prize-collecting traveling salesman and related problems

Written by: Czirják Lilla
Supervisor: Dr. Király Tamás
May, 2024

1 Introduction

The traveling salesman problem is today considered to be classic in combinatorial optimization. The aim of this problem is to find a shortest Hamiltonian cycle in a given complete graph on n nodes with metric edge weights. It is well known that this problem is NP-hard, but numerous approximate algorithms have been devised to solve it.

Many different generalizations of the problem have been formulated. The subject of this essay is the prize-collecting version of the traveling salesman problem, in which, in addition to the metric edge weights, a positive weighting of the vertices of the graph is given: the prizes that the salesman loose by missing out a city. In this case, we do not require that the tour contains all vertices, i.e. we are not looking for a Hamiltonian cycle, but for a cycle of arbitrary size on the vertices of the graph, starting from a given root vertex, for which the difference between the collected revenue and the travel cost is maximal. In many cases, however, it is more appropriate to think of the vertex weights as penalties paid for missed vertices rather than revenues. Thus, we can formalize the problem as a minimization problem.

During the semester, in addition to studying the literature, I worked on implementing a simpler heuristic approximation algorithm and testing it on random graphs, and tried some tour-improvement methods to find a better solution starting from a given solution.

2 The prize-collecting traveling salesman

Definition 1. *Be given a complete graph $G = (V,\ E)$, a root $r \in V$, $c_e \geq 0\ \forall e \in E$ metric edge lengths, and vertex weights $\pi_v \geq 0\ \forall v \in V\backslash\{r\}$.*
The prize-collecting traveling salesman problem is to find a cycle $C = (V_C,\ E_C)$ in G, so that $r \in V_C$, and $\sum_{e\in E_C} + \sum_{v\in V\backslash V_C}$ is minimal.

A linear programming relaxation of the problem can be formulated as follows:

$$min \sum_{e\in E} c_e x_e + \sum_{v\in V} \pi_v (1 - y_v)$$

$$\begin{array}{rcl} x(\delta(v)) & = & 2y_v \quad \forall v \in V\backslash\{r\} \\ x(\delta(r)) & \leq & 2 \\ x(\delta(S)) & \geq & 2y_v \quad \forall S \subseteq V\backslash\{r\},\ v \in S \\ y_r & = & 1 \\ x_e & \geq & 0 \quad \forall e \in E \\ y_v & \geq & 0 \quad \forall v \in V \end{array}$$

The variables x_e correspond to the edges, while the variables y_v can be matched to the vertices. Given $S \subseteq V$, $\delta(S) := \{e \in E : \ |e \cap S| = 1\}$, and $\delta(v) := \delta(\{v\})$.

The linear programming relaxation can be used to calculate a lower bound for the solution.

3 A heuristic algorithm

For heuristic algorithms, we usually cannot give an upper bound on the error of the solution compared to the optimum, because it can be very large when running the algorithm on extreme examples. In contrast, they work well in many cases on random examples and are therefore useful from a practical point of view.

Consider the following algorithm. Given an n-vertex complete graph with metric edge length, a positive weighting of the vertices, and a fixed root node.

Starting from the root as a one-point tour, add vertices one by one to the existing tour. At each step, we choose the best option to increase the tour, i.e., we try to insert all the outlying vertices between each pair of adjacent vertices in the current tour, and then perform the insertion that improves the solution value the most. - Here we also allow the improvement to be negative at a given step. - Finally, we choose the one that gives the best result from n tours of different lengths.

3.1 Tour-improvement methods

Once we know a solution - we can even begin from a random walk - we can try to improve it with additional heuristic algorithms. The algorithm described above has been tested with two improvement procedures.

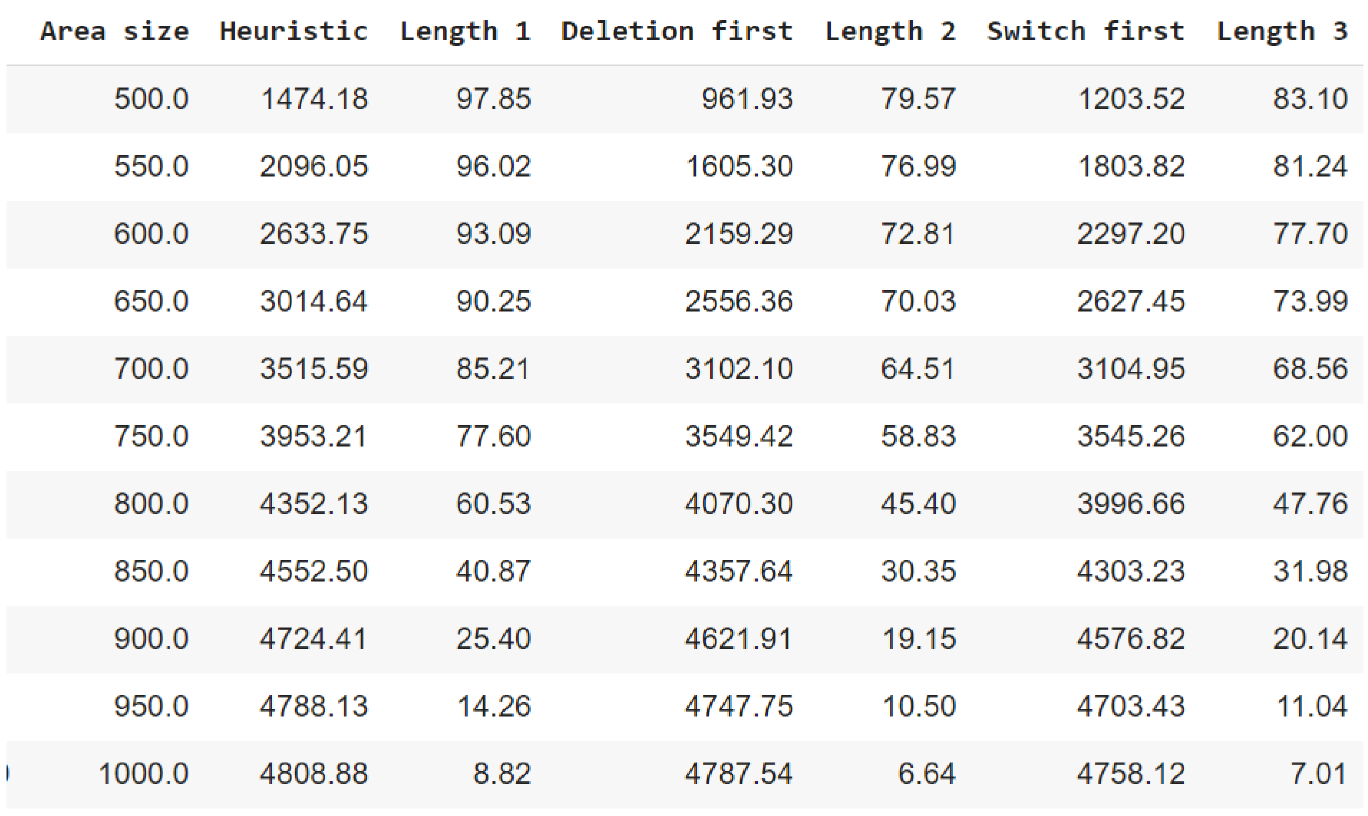
1. Eliminating vertices For a given tour, we can simply look at each vertex to see if leaving it out improves the value of the solution.

2. Switching edges For each pair of edges $(v_1v_2,\ u_1u_2)$ in the given tour (where the edge v_1v_2 is passed through sooner than u_1u_2), we check whether the solution can be improved as follows. The edges v_1v_2 and u_1u_2 are deleted from the solution and replaced by the edges v_1u_1 and v_2u_2. The path $v_2 - u_1$ is replaced by the path $u_1 - v_2$ (walked through from the opposite direction). Repeat this.

3.2 Testing

Testing the algorithms on random graphs generated with the following parameters. The number of vertices is fixed n=100 and the vertex weights are random integers between 0 and 100. To obtain the edge weights, I assigned uniformly random coordinates to the vertices and calculated Euclidean distances from these. I used integer values to speed up the run time. I rounded the distances upwards, as it is easy to see that this preserves the metric property.

I used the heuristic algorithm described earlier, and then ran the two correction methods mentioned above on the resulting runs. I compared how much improvement can be obtained on given examples by first applying the vertex elimination and then the edge swapping, and by applying in the reverse order. I was changing the size of the area from which the nodes are chosen, i.e. the length of the interval used to generate random coordinates. In the following table, each row stands for the average result of 100 runs with the given area size. According to the results, the heuristic algorithm can be improved significantly by the correction methods. Starting with the vertex elimination turned out to be more effective in case of small area sizes, however, it swaps with increasing the area size.



Area size	Heuristic	Length 1	Deletion first	Length 2	Switch first	Length 3
500.0	1474.18	97.85	961.93	79.57	1203.52	83.10
550.0	2096.05	96.02	1605.30	76.99	1803.82	81.24
600.0	2633.75	93.09	2159.29	72.81	2297.20	77.70
650.0	3014.64	90.25	2556.36	70.03	2627.45	73.99
700.0	3515.59	85.21	3102.10	64.51	3104.95	68.56
750.0	3953.21	77.60	3549.42	58.83	3545.26	62.00
800.0	4352.13	60.53	4070.30	45.40	3996.66	47.76
850.0	4552.50	40.87	4357.64	30.35	4303.23	31.98
900.0	4724.41	25.40	4621.91	19.15	4576.82	20.14
950.0	4788.13	14.26	4747.75	10.50	4703.43	11.04
1000.0	4808.88	8.82	4787.54	6.64	4758.12	7.01

The code is available at the following link:
https://colab.research.google.com/drive/1ODzrT0YPFiCL41IfABo3TVagoZoS9Jd1?usp=sharing

References

[1] Blauth, Klein, Nägele: A Better-Than-1.6-Approximation for Prize-Collecting TSP (2023)

[2] Goemans, Williamson: A general approximation technique for constrained forest problems (1995)

[3] Ausiello, Bonifaci, Leonardi, Marchetti-Spaccamela: Prize-Collecting Traveling Salesman and Related Problems

[4] Ahamdi, Gholami, Hajiaghayi, Jabbarzade, Mahdavi: 2-Approximation for Prize-Collecting Steiner Forest