Travelling Salesperson

Advanced Algorithms - Project 1
Round 2

Johanna Bromark, bromark@kth.se
Jacob Björkman, jacobbj@kth.se
Isabelle Hallman, ihal@kth.se

Final result: 31.3 on Kattis, ID: 2373477

Table of contents

1. Introduction

The goal of this report is to compare different approximation techniques for the traveling salesperson problem as given on Kattis (Austrin 2006), a portal for distributing and testing programming exercises used by KTH . Different algorithms were implemented in C++ and compared against each other.

1.1 The travelling Salesperson Problem

The travelling salesperson problem (TSP) consists of deciding the shortest route (by cost) to visit a set of cities exactly one time each and return to the origin city. This problem is a well known optimization problem that is NP-hard. The TSP problem can also be formulated to consider multiple salespersons (vehicle routing problem) or non symmetric cost for the paths between cities. This project is based on the instructions on Kattis where the problem is a symmetric euclidean distance TSP.

2. Method

2.1 Data structure

The following descriptions of data structures refer to those given in C++.

To simplify the handling of each city, each input city was made into a “town” object, implemented as a structure. They contained information about x and y coordinates and an index, as well as a method for calculating the distance to another city. All town objects were placed in a vector where the order of the cities in the tour was the order of the vector.

2.2 Creating tours

From the input data an initial tour was created through the greedy algorithm described below.

2.2.1 Greedy algorithm

Apart from a more naive approach where the towns are linked in the order they are inputted this is possibly the simplest and most straightforward method of creating a tour. From a random starting town the distance to every other town is calculated, and the town with the shortest distance is chosen as the next town to go to. From this next town the process is repeated until all towns are connected and the last town is connected to the first. As one can easily see this will not always result in a perfect shortest tour. The time complexity is $O(n^2)$, where n is number of cities.

2.3 Optimizing tours

After a tour is created there are different ways of optimizing it. Below are the methods used in this project.

2.3.1 2-opt

2-opt is a simple local search algorithm that will optimize a tour by checking if swapping four vertices’ edges, from A-B C-D to A-C B-D, will result in a shorter tour. If so, the change is accepted. There are two different versions of 2-opt. One version is a complete 2-opt local search where every possible swap for every edge is tested. The other is to swap directly when a shorter path is found, break the loop and try from the next original edge. In this project the first method is used. The time complexity for the 2-opt comparison is $O(n^2)$ where n is number of cities and the time complexity for the swap in our case is O(n).

2.3.2 2.5-opt

2.5-opt (also called 2H-opt) involves one more vertex and edge than 2-opt to try to get an even shorter path, without any added complexity. Here the nodes A-B-C and D-E will be swapped to A-C and D-B-E. In this project we also looked at the other alternative to involve another vertex which would make A-B and D-E-F into A-E-B and D-F. The time complexity for the 2-opt comparison is $O(n^2)$

where n is number of cities and the time complexity for the swap in our case is O(n).

2.3.3 Randomizing initial tour

To avoid local minima, the optimizations can be run on different initial tours. The initial tour is randomized, put through the other algorithms, and then the result is saved. This is repeated and if the new result is better, it is saved. In this particular problem there was a time limit of two seconds, so the above strategy was repeated until the time limit was reached.

2.4 Combining algorithms

During the project it was found that different combinations of algorithms gave a better result.

We tested:

1. Greedy
2. Greedy + 2-opt
3. Greedy + 2-opt + 2.5-opt
4. Greedy + 2-opt + 2.5-opt Repeated twice
5. Greedy + 2-opt + 2.5-opt + Randomization

2.5 Testing

Two methods of testing were used to compare the different algorithms. One was the built in test on Kattis which run 50 tests of different input data with a maximum of 1000 cities and gave a score based on how good the outputted route was. These tests are unknown to the user. Kattis also had a time limit of two seconds.

The other tests were conducted with known test data (see 2.5.1 below).

2.5.1 Test data

The test data was gathered from TSPLIB (Reinelt, N.D.). More specifically three data sets A, B and C of 38, 194 and 734 cities respectively (cities in Djibouti, Qatar and Uruguay). The given input data was the number of cities followed by x and y coordinates for each city. The data was inputted into the program by standard input and placed in the data structure described in 2.1.

3. Results

The results for the test cases, as described in section 2.5.1, are shown in Figure 1 below. The y-axis describes the calculation of the tour length of the result given by the algorithm, divided by the length of the optimal tour. The greedy algorithm by itself did not perform that well and a smaller set of cities seemed to make it perform even worse. Combining it with 2-opt resulted in a much better route. Adding 2.5-opt did not make much of a difference for set A and B but improved the route for the larger set C. The best performance overall was achieved when random initial paths were used. For set A this combination resulted in the optimal tour, and the result for set B was also improved. The tour for the large dataset C was not improved significantly.

The results from Kattis are shown in Figure 2 below. The y-axis describes the score on Kattis. In line with the test data the combination of Greedy, 2-opt and 2.5-opt gave the best route. A difference from the tests in figure 1 is that a fourth test case, running Greedy + 2-opt + 2.5-opt twice, was added which gave a significantly better result. But similarly to the test cases above, the best combination was when randomization was introduced.

It also became apparent that 2-opt and 2.5-opt could not be used by themselves, when considering the input order as an initial tour. They were not efficient enough by themselves

to pass the time limit of two seconds.

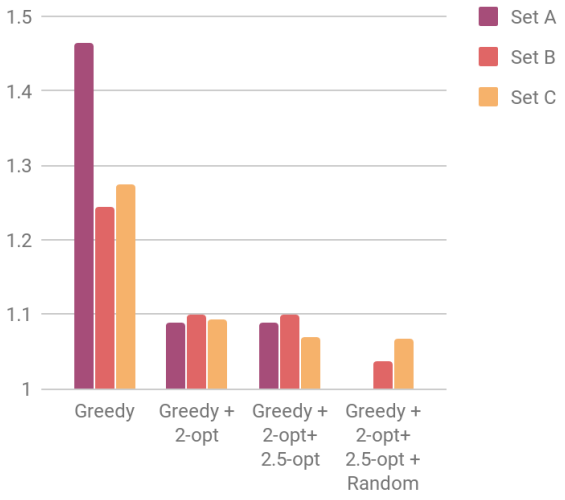

Figure 1: Comparing result from the optimal tour in the test data (result/optimal)

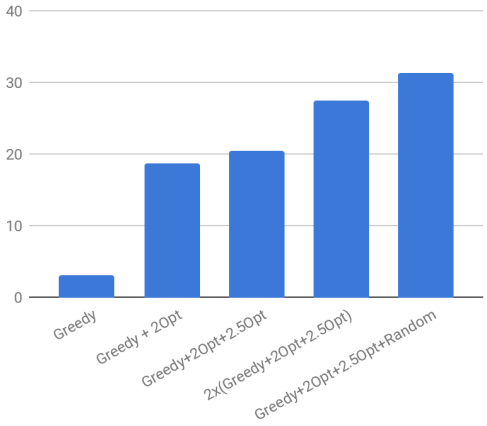

Figure 2: Kattis results for the different combinations

4. Conclusion

As the result shows, by adding local minimization a much better route is found (see figure 1).

The greedy algorithm by itself did not give a satisfactory result but combined with 2-opt the result improved significantly. Therefore 2.5-opt was added as well, which increased the score a bit more (see figure 1). The most successful method turned out to be using initial random tours and then Greedy + 2-opt + 2.5-opt, which yielded a result of about 31.3 points on Kattis (see figure 1). Due to the randomization, local minimas were avoided which is why a better result was achieved.

It would be interesting to check the impact of using a different algorithm for generating the initial tour (here, we only used Greedy) - for example Clarke-Wright of Christofides. It would also be interesting to see how some other optimization algorithm would affect the outcome, for example using 3-opt. Unfortunately we didn't have time to implement said algorithms this time around.

5. References

Austrin, Per. 2006. "Travelling Salesperson 2D." *KTH CSC Avancerade Algoritmer*. 2017-11-05 https://kth.kattis.com/problems/tsp

Reinelt, Gerhard. n.d. "TSPLIB." *Universität Heidelberg Institut Für Informatik. 2017-11-05 http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/*