Algorithm Engineering
WS 22/23

Donatien Leray
Cedric Wiese

# A Comparative Study of nearest-neighbor heuristic, the 2-Opt heuristic using the nearest-neighbor tour or the initial tour as start

## Abstract

We implement the nearest-neighbor heuristic, as well as the 2-opt heuristic for finding optimized tours in graphs. We first explain both algorithms in detail, and explain the results according to the country samples provided by www.math.uwaterloo.ca.

## 1 Introduction

The Traveling Salesman Problem is one of the most significant problems in computer science. Finding the smallest possible tours in graphs, where each instance is exactly visited once, is necessary for a vast variety of applications and usages.

## 2 Preliminaries

The nearest-neighbor algorithm is a good start for finding a tour, which is, in most cases, significantly shorter than a tour which has been randomly chosen. In combination with a 2-opt heuristic, tours can effectively be improved. Although it is hardly possible to find an ideal tour, the results can be fairly close to the optimum.

### 2.1 Nearest-Neighbor Heuristic

The nearest-neighbor algorithm works by starting at a (potentially random) city, and finding the next closest city, visiting the closest city next. From the new city, a new search for the next closest city begins, which again is visited. This is repeated, until no more cities can be visited. In this case, the tour has to go back to the starting point. The result is a tour, in which each city has been visited exactly one time, by always visiting the next closest city.

### 2.2 2-opt heuristic

The 2-Opt heuristic works by brute forcing edge swaps. Starting off with a tour, the algorithm brute forces all possible edge swaps, checking whether the resulting tour is shorter, equal, or longer. In case of the tour being shorter, the tour is updated, and the process starts again. The algorithm stops, when no edge swaps can be performed, which could reduce the tour length. It should be noted, that the algorithm stopping, does not result in an ideal tour. Just because no improving edge swaps are possible, does not mean that the tour has the optimal length

## 3 Algorithm & Implementation

### 3.1 Nearest-Neighbor Heuristic

The implementation of the nearest-neighbor heuristic is straight forward. We start off with a list of cities, each containing a unique ID, as well as 2D coordinates, consisting of a longitude and latitude value. In our implementation, we made sure that only distinct cities exist in our List. We then start off at the first city, calculating the distance between the first city and all other cities. Starting with a maximum value, we always update the distance, when a smaller one has been found. Additionally, in case we find a new smaller distance, we also save the position of the according city. This results in knowing which city is the closest to the first city. Next, we perform a simple swap, by swapping the closest city with the second city. We then have the first two cities in correct order, and can now continue with the second city, repeating the above steps. While calculating the nearest-neighbor tour, we always have list-slice, which is already a correct nearest-neighbor path, and an unsorted list slice. The sorted slice grows, until the whole list represents a nearest-neighbor path.

### 3.2 2-Opt Heuristic

For the 2-Opt Heuristic, we start with a tour, which the algorithm should optimize.

We also initialize a variable "Improved", which tracks, whether an edge swap was possible to reduce the tour length. While we were able to find a positive edge swap, we continuously run a double for loop, brute forcing all possible edge swaps. Before the two for-loops start, we set "Improved" to false. It will only be set to true again, if an edge swap is made.

The first for loop goes through the edges of the list of cities, so does the second for loop as well. While the index of the outer for loop (here: i) takes an edge by taking a city at position i and i+1, the index of the inner for-loop takes an edge by taking a city at position j and j+1. The algorithm now checks a potential edge swap, resulting in two new edges. One edge between city i and j, and another edge between city i+1 and j+1. Since the rest of the tour is irrelevant, no computing has to be done for the tour before, after or in between the edges. One can then compare the length of the original edges (i $\rightarrow$ i+1 and j $\rightarrow$ j+1) to the length of the new edges (i $\rightarrow$ j and i+1 $\rightarrow$ j+1). If the new length is smaller than the old length, the edge swap is performed on the list, as well as setting the "Improved" variable to true.

When at some point no edge swaps can be performed in order to reduce the length of the tour, the while loop will end. If edge swaps were performed, we now have a shorter tour than before.

### 3.3 AE-Principles and techniques

Regarding the AE-principles, we strongly relied on the AE-cycle. We started off by creating an abstract idea of how our code could look like. We then started to implement it step by step, fixing it along the way until it worked as expected. We then tested the run time on various instances, figuring out that we do not need to compute the whole tour every time, so only computed the difference in length for two edges.

This significantly improved the running time.

As for libraries, we used BufferedReader/-Writer and Filereader/-Writer in order to efficiently read and write our hard drive.

## 4 Experimental Evaluation

In this section, we evaluate our algorithms on the official TSP data sets based on the real world. For the first part of the experiment, we run nearest-neighbor heuristic on all 25 national TSPs. Then proceed to run the 2-Opt heuristic on these instances using once the initial tour as start tour and once the tour produced by the nearest-neighbor-algoritm.

### 4.1 Data and Hardware

All the algorithm were processed by a single core of an Intel i7-2600 CPU with 3.80GHz on a machine with 16GB RAM during the Experiments. The Experiment were conducted on the 25 National TSP datasets you can find on https://www.math.uwaterloo.ca/tsp/world/countries.html. The TSPs were derived from data contained in the National Imagery and Mapping Agency database of geographic feature names.

### 4.2 Experimental Results

#### 4.2.1 nearest-neighbor heuristic on all 25 national TSPs

In this Experiment, we ran the nearest-neighbor heuristic on all 25 inputs, and measure (per instance) the running time as well as the resulting tour length. Compare the latter to the lower bound for each instance provided here: http://www.math.uwaterloo.ca/tsp/world/summary.html by computing the ratio of the tour length you computed and the value in the 'Bound' column. All the results shown in detail 1. How the runtime and ratio changed with the input size is graphically represented in 1.

#### 4.2.2 2-Opt heuristic on all initial 25 national TSPs

We then proceed to run the 2-Opt heuristic on the initial graphs from www.math.uwaterloo.ca/tsp/world/countries.html. the collected data is listed in 2.

#### 4.2.3 2-Opt heuristic on the tours produced with the nearest-neighbor heuristic

in this part of the experiment we again run the 2-Opt heuristic, but this time we use as start path the resulting path from the nearest-neighbor heuristic. the collected data is listed in 3.

after completing the experiment 4.2.2 and 4.2.3 we compare the results of both and graphically represented the result in 2.

#### 4.2.4 Visualization of the tours for Djibouti, Qatar and Luxembourg

For the instances Djibouti, Qatar and Luxembourg we provided the images of the following tours: (a) the tour produced by the nearest-neighbor heuristic, (b) the initial tour for the 2-Opt heuristic specified by the order of the points in the file, (c) the intermediate tour after (roughly) half of the total number of edge swaps were performed, (d) the final tour produced by the 2-Opt heuristic when starting with the tour from (b), (e) the final tour produced by the 2-Opt heuristic when starting with the tour from (a).

## 5 Discussion and Conclusion

In this work, we reviewed and compared the nearest-neighbor heuristic, the 2-Opt heuristic. We showed that the 2-Opt heuristic has outperforms the nearest-neighbor algorithm in solving the traveling salesman problem (TSP) in real-world scenarios.

This is because the 2-Opt algorithm is able to make larger improvements to the solution by swapping multiple edges at once, whereas the nearest-neighbor algorithm only makes local improvements by adding the closest city to the tour at each step. Additionally, the 2-Opt algorithm is able to escape local optima and find better solutions by considering a wider range of possible swaps. This makes it a more powerful and efficient algorithm for solving TSP in real-world scenarios.

Furthermore we compared the 2-Opt heuristic using the nearest-neighbor tour or the initial tour as start for the TSP. We could observe that the 2-Opt heuristic using the nearest-neighbor tour always outperformed the 2-Opt heuristic using the initial tour.

It is not surprising that the 2-Opt heuristic algorithm performs better when using a nearest-neighbor tour as the initial tour compared to a random initial tour. This is because the nearest-neighbor tour is a good starting point for the 2-Opt heuristic to work on because it usually is a good approximation of an optimal solution, and it already has some structure that the 2-Opt heuristic can improve upon.

On the other hand, a random initial tour has no inherent structure or quality, making it a less favorable starting point for the 2-Opt heuristic. The 2-Opt heuristic is a local search algorithm that relies on the quality of the initial tour to explore the solution space effectively. With a random initial tour, the 2-Opt heuristic may need to perform many iterations to improve the solution and escape a suboptimal local optimum.

Therefore, using a nearest-neighbor tour as the initial tour for the 2-Opt heuristic provides a good starting point for the algorithm to work on, leading to better performance and more efficient exploration of the solution space.

| Country | Cities | rutime in ms | length in m | Bound in m | ratio length to Bound in % |
|---|---|---|---|---|---|
| Argentina | 9.152 | 50,88 | 1.052.616 | 837.479 | 20,44 |
| Burma | 33.708 | 1.391,42 | 1.190.440 | 959.011 | 19,44 |
| China | 71.009 | 6.088,10 | 5.671.474 | 4.565.452 | 19,50 |
| Djibouti | 38 | <0,01 | 9.749 | 6.656 | 31,73 |
| Egypt | 7.146 | 81,55 | 259.165 | 172.386 | 33,49 |
| Finland | 10.639 | 131,08 | 651.055 | 520.527 | 20,05 |
| Greece | 9.882 | 125,25 | 386.775 | 300.899 | 22,20 |
| Honduras | 14.473 | 57,54 | 224.460 | 177.092 | 21,10 |
| Ireland | 8.246 | 81,55 | 259.165 | 206.171 | 20,45 |
| Japan | 9.847 | 142,81 | 639.253 | 491.924 | 23,05 |
| Kazakhstan | 9.976 | 113,39 | 1.355.830 | 1.061.881 | 21,68 |
| Luxembourg | 980 | 0,49 | 14.213 | 11.340 | 20,21 |
| Morocco | 14.185 | 237,97 | 531.000 | 427.377 | 19,51 |
| Nicaragua | 3.496 | 6,12 | 118.963 | 96.132 | 19,19 |
| Oman | 1.979 | 6,80 | 120.424 | 86.891 | 27,85 |
| Panama | 8.079 | 27,48 | 146.580 | 114.855 | 21,64 |
| Qatar | 194 | 0,04 | 11.893 | 9.352 | 21,37 |
| Rwanda | 1.621 | 0,79 | 32.141 | 26.051 | 18,95 |
| Sweden | 24.978 | 710,87 | 1.079.235 | 855.597 | 20,72 |
| Tanzania | 6.117 | 46,46 | 502.244 | 394.718 | 21,41 |
| Uruguay | 734 | 0,60 | 100.692 | 79.114 | 21,43 |
| Vietnam | 22.775 | 603,98 | 716.438 | 569.288 | 20,54 |
| Western Sahara | 29 | <0,01 | 36.388 | 27.603 | 24,14 |
| Yemen | 7.663 | 70,11 | 303.470 | 238.314 | 24,47 |
| Zimbabwe | 929 | 1,73 | 122.528 | 95.345 | 22,19 |

Table 1: all collected data from nearest-neighbor heuristic on all 25 national TSPs sorted alphabetical by country name

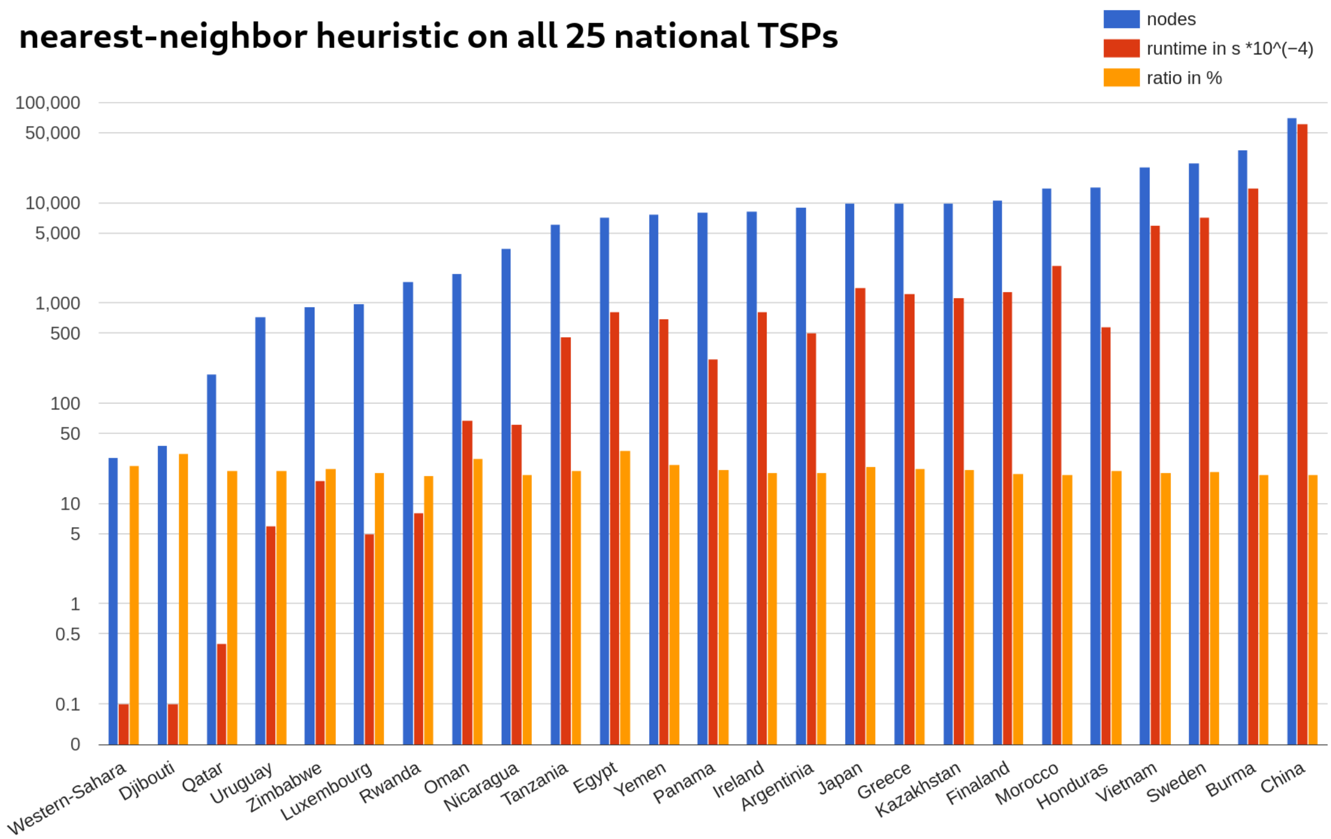


Figure 1: The nearest-neighbor heuristic on all 25 national TSPs sorted by increasing number of cities. Note the log scale on the horizontal axis.

| Country | Cities | rutime in s | length in m | Bound in m | ratio length to Bound in % | swaps |
|---|---|---|---|---|---|---|
| Argentina | 9.152 | 2.31 | 976.148 | 837.479 | 16,56 | 53231 |
| Burma | 33.708 | 65.68 | 1.120.774 | 959.011 | 16,87 | 120332 |
| China | 71.009 | 343.99 | 5.325.716 | 4.565.452 | 16,65 | 379554 |
| Djibouti | 38 | <0,01 | 9.069 | 6.656 | 36,25 | 66 |
| Egypt | 7.146 | 3.03 | 201.074 | 172.386 | 16,64 | 38466 |
| Finland | 10.639 | 5.62 | 613.262 | 520.527 | 17,82 | 45177 |
| Greece | 9.882 | 5.23 | 351.598 | 300.899 | 16,85 | 34629 |
| Honduras | 14.473 | 2.40 | 204.642 | 177.092 | 15,56 | 52342 |
| Ireland | 8.246 | 3.31 | 242.144 | 206.171 | 17,45 | 40100 |
| Japan | 9.847 | 6.07 | 578.363 | 491.924 | 17,57 | 42561 |
| Kazakhstan | 9.976 | 5.83 | 1.239.380 | 1.061.881 | 16,71 | 50017 |
| Luxembourg | 980 | 0.01 | 13.244 | 11.340 | 16,79 | 3374 |
| Morocco | 14.185 | 9.07 | 503.792 | 427.377 | 17,88 | 56047 |
| Nicaragua | 3.496 | 0.25 | 114.343 | 96.132 | 18,94 | 14998 |
| Oman | 1.979 | 0.22 | 99.348 | 86.891 | 14,34 | 8120 |
| Panama | 8.079 | 1.45 | 132.043 | 114.855 | 14,96 | 34132 |
| Qatar | 194 | <0,01 | 10.499 | 9.352 | 12,26 | 493 |
| Rwanda | 1.621 | 0.03 | 29.951 | 26.051 | 14,97 | 4494 |
| Sweden | 24.978 | 30.96 | 1.007.925 | 855.597 | 17,80 | 93689 |
| Tanzania | 6.117 | 2.11 | 464.038 | 394.718 | 17,56 | 24596 |
| Uruguay | 734 | 0.02 | 93.593 | 79.114 | 18,30 | 2452 |
| Vietnam | 22.775 | 25.97 | 660.941 | 569.288 | 16,10 | 74486 |
| Western Sahara | 29 | <0,01 | 28.102 | 27.603 | 1,81 | 49 |
| Yemen | 7.663 | 2.96 | 280.618 | 238.314 | 17,75 | 37957 |
| Zimbabwe | 929 | 0.05 | 111.771 | 95.345 | 17,23 | 2986 |

Table 2: all collected data from 2-Opt heuristic on all initial 25 national TSPs sorted alphabetical by country name

| Country | Cities | rutime in s | length in m | Bound in m | ratio length to Bound in % | swaps |
|---|---|---|---|---|---|---|
| Argentina | 9.152 | 1,98 | 935.257 | 837.479 | 11,68 | 4469 |
| Burma | 33.708 | 49,26 | 1.063.827 | 959.011 | 10,93 | 18826 |
| China | 71.009 | 269,95 | 5.042.742 | 4.565.452 | 10,45 | 43921 |
| Djibouti | 38 | <0,01 | 7.448 | 6.656 | 11,90 | 54 |
| Egypt | 7.146 | 2,48 | 191.200 | 172.386 | 10,91 | 6470 |
| Finland | 10.639 | 4,75 | 577.586 | 520.527 | 10,96 | 6645 |
| Greece | 9.882 | 4,30 | 333.354 | 300.899 | 10,79 | 7357 |
| Honduras | 14.473 | 1,84 | 195.750 | 177.092 | 10,54 | 4338 |
| Ireland | 8.246 | 3,63 | 228.513 | 206.171 | 10,84 | 5712 |
| Japan | 9.847 | 4,30 | 548.611 | 491.924 | 11,52 | 8970 |
| Kazakhstan | 9.976 | 4,41 | 1.192.014 | 1.061.881 | 12,255 | 8041 |
| Luxembourg | 980 | 0,01 | 12.617 | 11.340 | 11,26 | 419 |
| Morocco | 14.185 | 8,16 | 475.098 | 427.377 | 11,17 | 8591 |
| Nicaragua | 3.496 | 0,25 | 106.139 | 96.132 | 10,41 | 1416 |
| Oman | 1.979 | 0,16 | 97.976 | 86.891 | 12,76 | 1883 |
| Panama | 8.079 | 0,88 | 127.119 | 114.855 | 10,68 | 3312 |
| Qatar | 194 | <0,01 | 10.102 | 9.352 | 8,02 | 140 |
| Rwanda | 1.621 | 0,02 | 28.947 | 26.051 | 11,12 | 499 |
| Sweden | 24.978 | 30,60 | 955.552 | 855.597 | 11,68 | 17307 |
| Tanzania | 6.117 | 1,72 | 437.963 | 394.718 | 10,96 | 4656 |
| Uruguay | 734 | 0,02 | 87.264 | 79.114 | 10,30 | 563 |
| Vietnam | 22.775 | 27,97 | 631.053 | 569.288 | 10,85 | 14698 |
| Western Sahara | 29 | <0,01 | 29.209 | 27.603 | 5,82 | 17 |
| Yemen | 7.663 | 2,38 | 266.144 | 238.314 | 11,68 | 6357 |
| Zimbabwe | 929 | 0,03 | 105.999 | 95.345 | 11,17 | 735 |

Table 3: all collected data from 2-Opt heuristic on the tours produced with the nearest-neighbor heuristic sorted alphabetical by country name

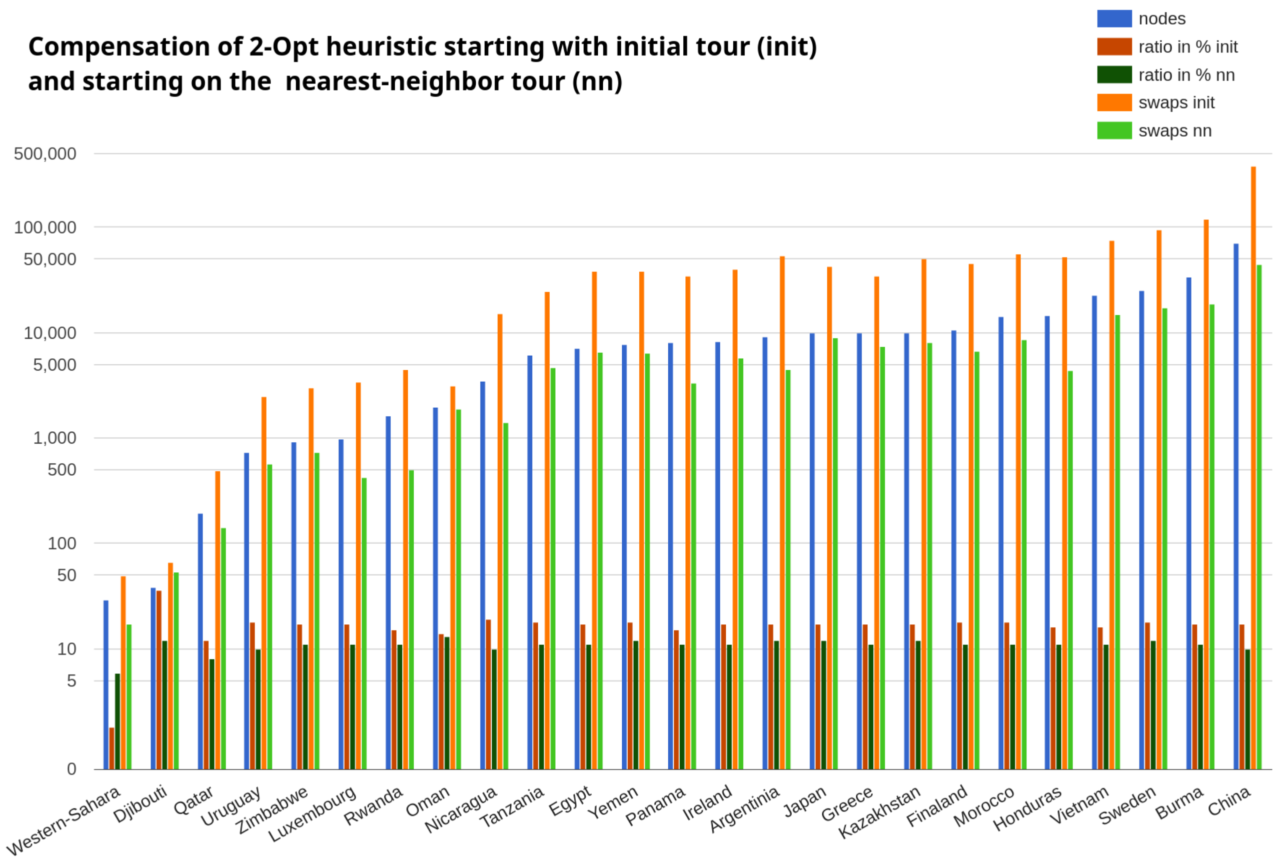


Figure 2: Compensation of 2-Opt heuristic starting with initial tour (init) and starting on the nearest-neighbor tour (nn) by increasing number of cities. Note the log scale on the horizontal axis.

## Djibouti

(a)

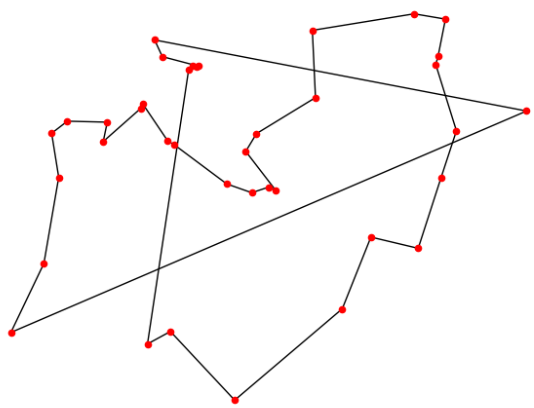

Figure 3: the tour produced by the nearest-neighbor heuristic for Djibouti.

(b)

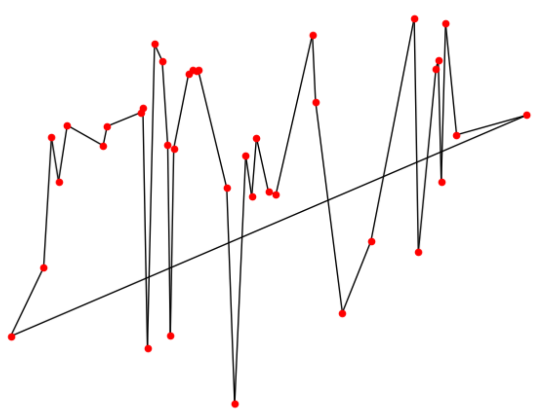

Figure 4: the initial tour for the 2-Opt heuristic specified by the order of the points in the file for Djibouti.

(c)

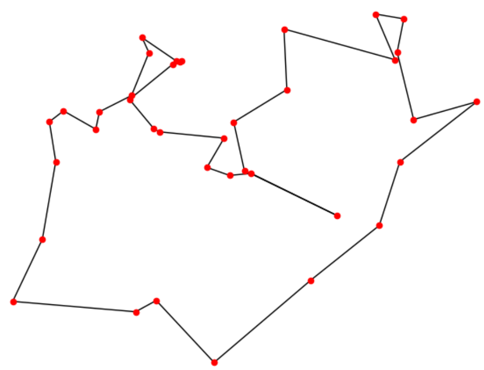

Figure 5: the intermediate tour after (roughly) half of the total number of edge swaps were performed for Djibouti.

(d)

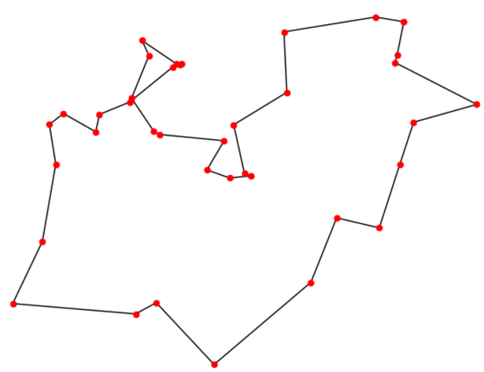

Figure 6: the final tour produced by the 2-Opt heuristic when starting with the tour from 4, for Djibouti.

(e)

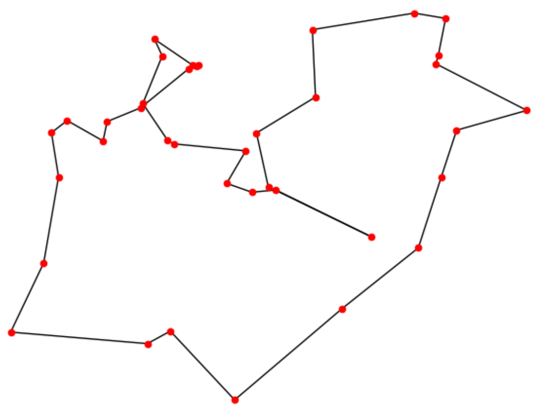

Figure 7: the final tour produced by the 2-Opt heuristic when starting with the tour from 3 for Djibouti.

## Qatar

(a)

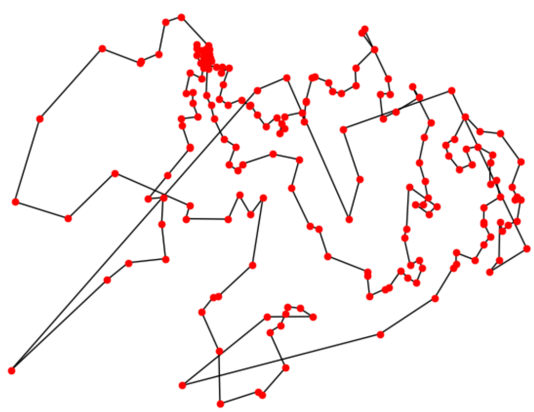

Figure 8: the tour produced by the nearest-neighbor heuristic for Qatar.

(b)

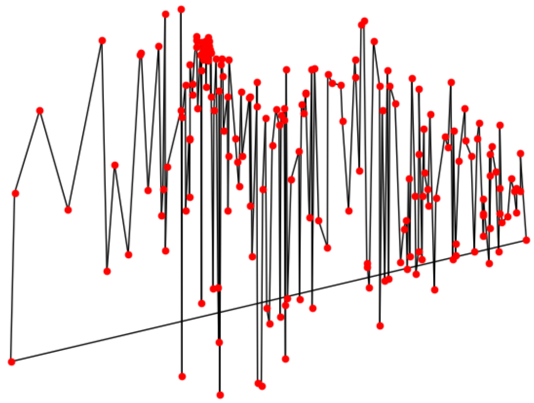

Figure 9: the initial tour for the 2-Opt heuristic specified by the order of the points in the file for Qatar.

(c)

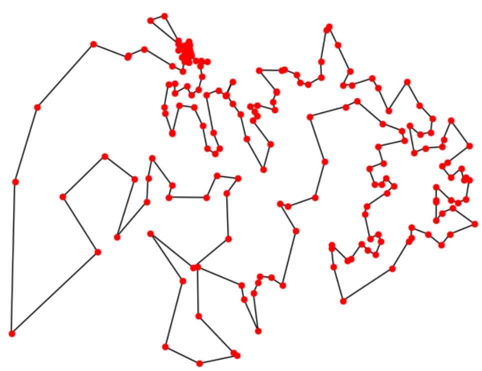

Figure 10: the intermediate tour after (roughly) half of the total number of edge swaps were performed for Qatar.

(d)

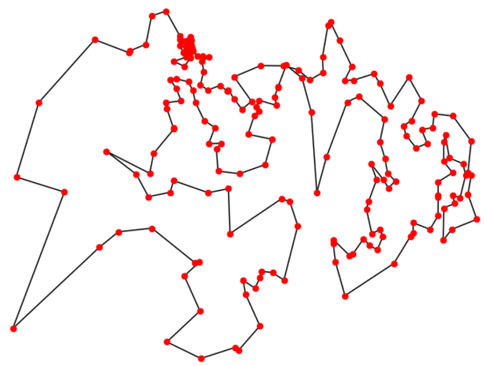

Figure 11: the final tour produced by the 2-Opt heuristic when starting with the tour from 9, for Qatar.

(e)

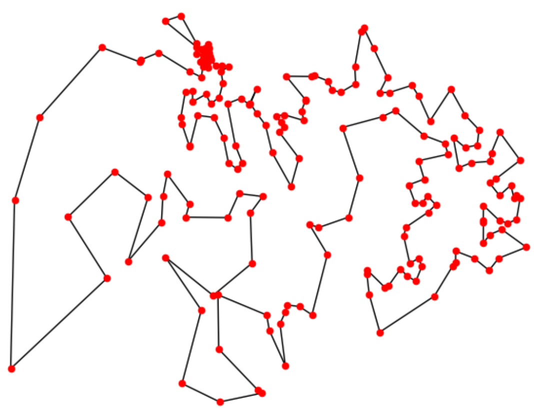

Figure 12: the final tour produced by the 2-Opt heuristic when starting with the tour from 8 for Qatar.

## Luxembourg

(a)

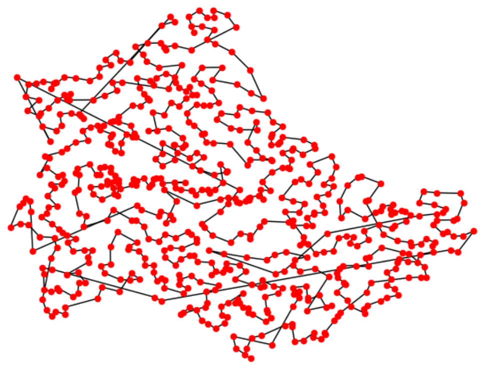

Figure 13: the tour produced by the nearest-neighbor heuristic for Luxembourg.

(b)

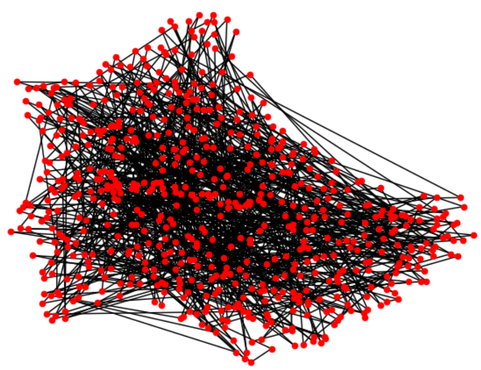

Figure 14: the initial tour for the 2-Opt heuristic specified by the order of the points in the file for Luxembourg.

(c)

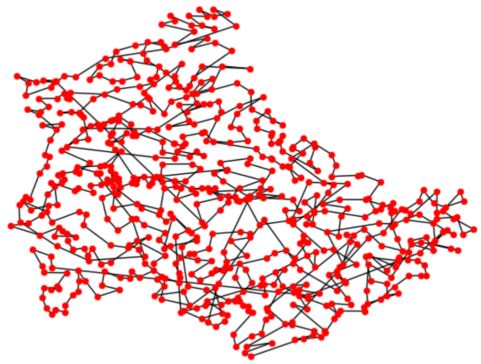

Figure 15: the intermediate tour after (roughly) half of the total number of edge swaps were performed for Luxembourg.

(d)

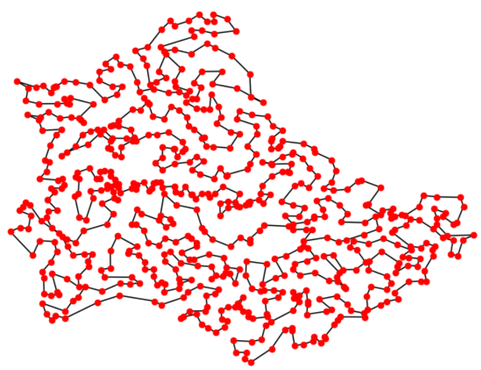

Figure 16: the final tour produced by the 2-Opt heuristic when starting with the tour from 14, for Luxembourg.

(e)

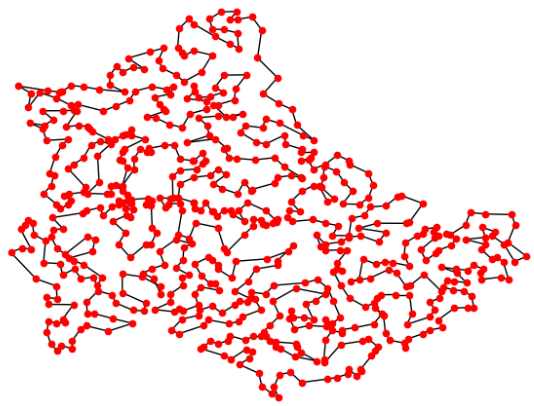

Figure 17: the final tour produced by the 2-Opt heuristic when starting with the tour from 13 for Luxembourg.