assignment1

October 14, 2024

1 Greedy heuristics

1.1 Description of the problem

We are given three columns of integers with a row for each node. The first two columns contain x and y coordinates of the node positions in a plane. The third column contains node costs. The goal is to select exactly 50% of the nodes (if the number of nodes is odd we round the number of nodes to be selected up) and form a Hamiltonian cycle (closed path) through this set of nodes such that the sum of the total length of the path plus the total cost of the selected nodes is minimized.

The distances between nodes are calculated as Euclidean distances rounded mathematically to integer values. The distance matrix should be calculated just after reading an instance and then only the distance matrix (no nodes coordinates) should be accessed by optimization methods to allow instances defined only by distance matrices.

1.2 Pseudocode for all implemented algorithms

1.2.1 Random solution

```
choose randomly without replacement 0.5 * n numbers from 0 to n
```

1.2.2 Nearest neighbor considering adding the node only at the end of the current path (NNHead)

The program needs the following arguments: - D - distance matrix (after adding weights to corresponding columns), loops are set to be inifinites - starting - first node from which we continue solution At the end of the program solution is in the variable: sol

Pseudocode

```
current = starting
sol = [starting]
V = {starting}
iterate 0.5 * len(D) - 1 times
    nn = argmin {D[current][x] where x not in V}
    sol.append(nn)
    add nn to V
    current = nn
```

1.2.3 Nearest neighbor considering adding the node at all possible position, i.e. at the end, at the beginning, or at any place inside the current path (NNWhole)

Pseudocode

```
current = starting
sol = [starting]
V = {starting}
iterate 0.5 * len(D) - 1 times
    best_dist = inf
    for j from 0 to len(solution) do
        nn = argmin {D[solution[j]][x] where x not in V}
        dist = D[solution[j]][nn]
        if dist < best_dist do
            best_dist = dist
            best_posj = j
            best_nn = nn
    insert best_nn to solution at position best_posi
    add n to V
```

1.2.4 Greedy cycle

Pseudocode

```
current = starting
sol = [starting]
V = {starting}
iterate 0.5 * len(D) - 1 times
    best_delta = inf
    for i from 0 to len(solution) - 1 do
        for j in NV:
            delta = D[sol[i], j] + D[j, sol[i+1]] - D[sol[i], sol[i + 1]]
            if delta < best_delta do
                best_i = i
                best_j = j
                best_delta = delta
    insert best_j to solution at position best_i
    remove best_j from NV
```

1.3 Results of a computational experiment: for each instance and method min, max and average value of the objective function.

[2]:
```
import pandas as pd
df = pd.read_csv('../results/assignment1.csv', sep=";")
grouped = df.groupby(['filename', 'solver'])
best_results = grouped['score'].agg(['min', 'max', 'mean'])
best_results
```

[2]:

filename	solver	min	max	mean
TSPA.csv	GreedyCycle	71488.0	74924.0	72730.520
	NNHead	83182.0	89433.0	85108.510
	NNWhole	78956.0	82916.0	81062.495
	RandomSolver	247495.0	287353.0	263815.545
TSPB.csv	GreedyCycle	48765.0	57294.0	51457.885
	NNHead	52319.0	59030.0	54390.430
	NNWhole	52992.0	57460.0	55024.635
	RandomSolver	185229.0	236136.0	212959.675

1.4 2D visualization of the best solution for each instance and method. Cost of nodes should be presented e.g. by a color, greyscale, or size.

[3]:
```
from tsp import TSP
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [16, 8]

best = df.loc[grouped['score'].idxmin()]
best
```

[3]:

	filename	solver	score	starting_node	solution
717	TSPA.csv	GreedyCycle	71488.0	117	[117,0,46,68,139,193,41,115,5,42,181,159,69,10…
324	TSPA.csv	NNHead	83182.0	124	[124,94,63,53,180,154,135,123,65,116,59,115,13…
518	TSPA.csv	NNWhole	78956.0	118	[115,68,46,139,41,108,18,146,22,159,193,34,184…
12	TSPA.csv	RandomSolver	247495.0	-1	[64,154,89,86,94,83,127,186,84,140,135,150,194…
1480	TSPB.csv	GreedyCycle	48765.0	80	[80,162,175,78,142,36,61,91,141,97,187,165,127…
1016	TSPB.csv	NNHead	52319.0	16	[16,1,117,31,54,193,190,80,175,5,177,36,61,141…
1210	TSPB.csv	NNWhole	52992.0	10	[107,40,63,135,54,113,179,66,94,47,60,148,4,14…
832	TSPB.csv	RandomSolver	185229.0	-1	[177,160,162,25,63,167,154,37,12,105,82,184,64…

[4]:
```
instance_a = TSP.from_csv('../data/TSPA.csv')
instance_b = TSP.from_csv('../data/TSPB.csv')
```

1.5 The best solutions for each instance and method presented as a list of nodes indices (starting from 0).

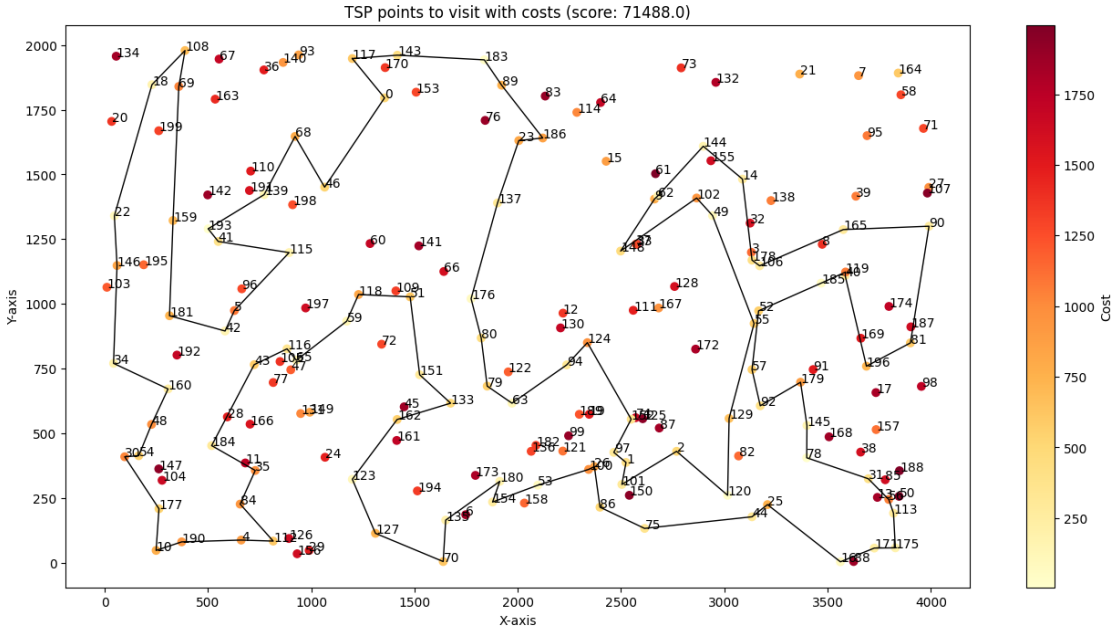
[5]:
```
def rotate_to_zero(solution):
    try:
        zeroi = solution.index(0)
        return solution[zeroi:] + solution[:zeroi]
    except:
        return solution
```

1.5.1 Results

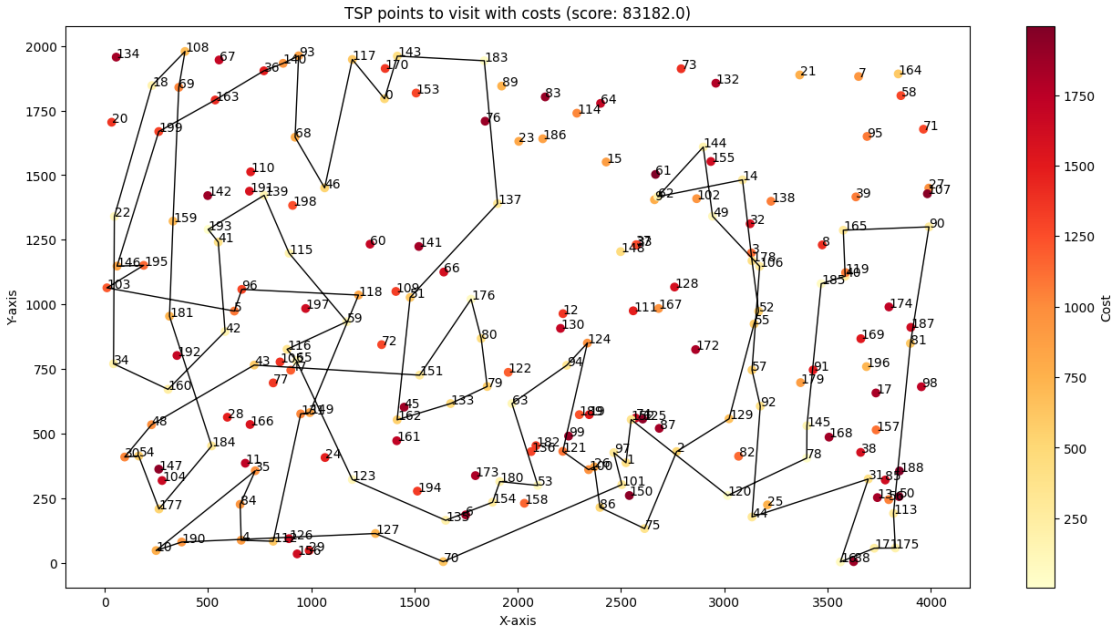
[6]:
```
for row in best.itertuples():

    print("----------", row.filename, row.solver)
    instance = instance_a
    if row.filename == 'TSPB.csv':
        instance = instance_b
    print(rotate_to_zero(eval(row.solution)))
    instance.visualize(eval(row.solution))
    print("------- Solution end -------")
```

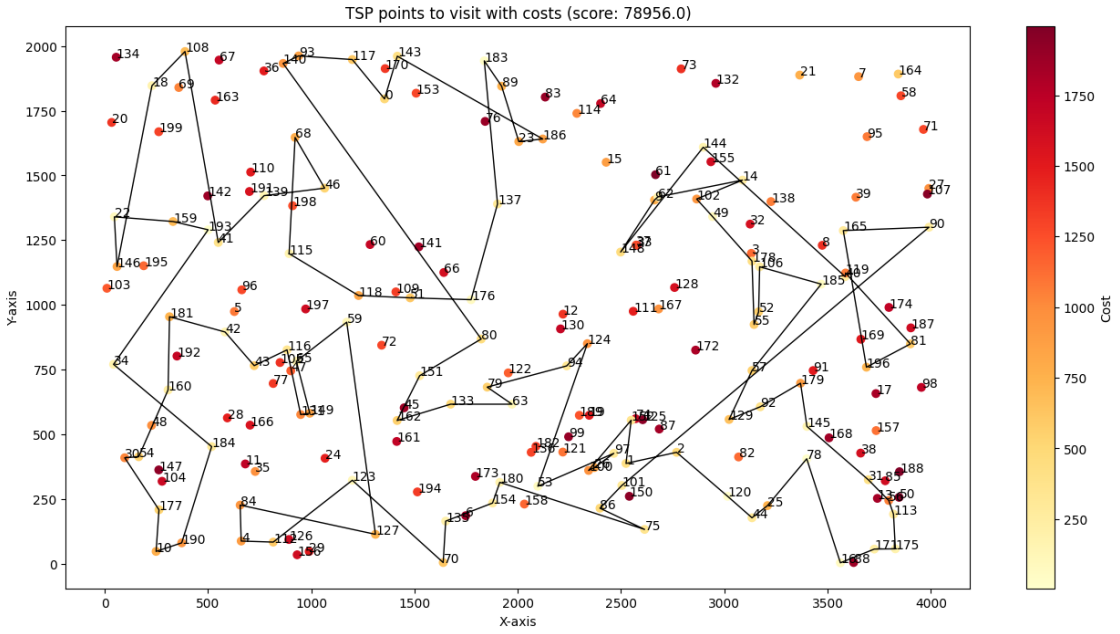
```
---------- TSPA.csv GreedyCycle
[0, 46, 68, 139, 193, 41, 115, 5, 42, 181, 159, 69, 108, 18, 22, 146, 34, 160,
48, 54, 30, 177, 10, 190, 4, 112, 84, 35, 184, 43, 116, 65, 59, 118, 51, 151,
133, 162, 123, 127, 70, 135, 180, 154, 53, 100, 26, 86, 75, 44, 25, 16, 171,
175, 113, 56, 31, 78, 145, 179, 92, 57, 52, 185, 119, 40, 196, 81, 90, 165, 106,
178, 14, 144, 62, 9, 148, 102, 49, 55, 129, 120, 2, 101, 1, 97, 152, 124, 94,
63, 79, 80, 176, 137, 23, 186, 89, 183, 143, 117]
```



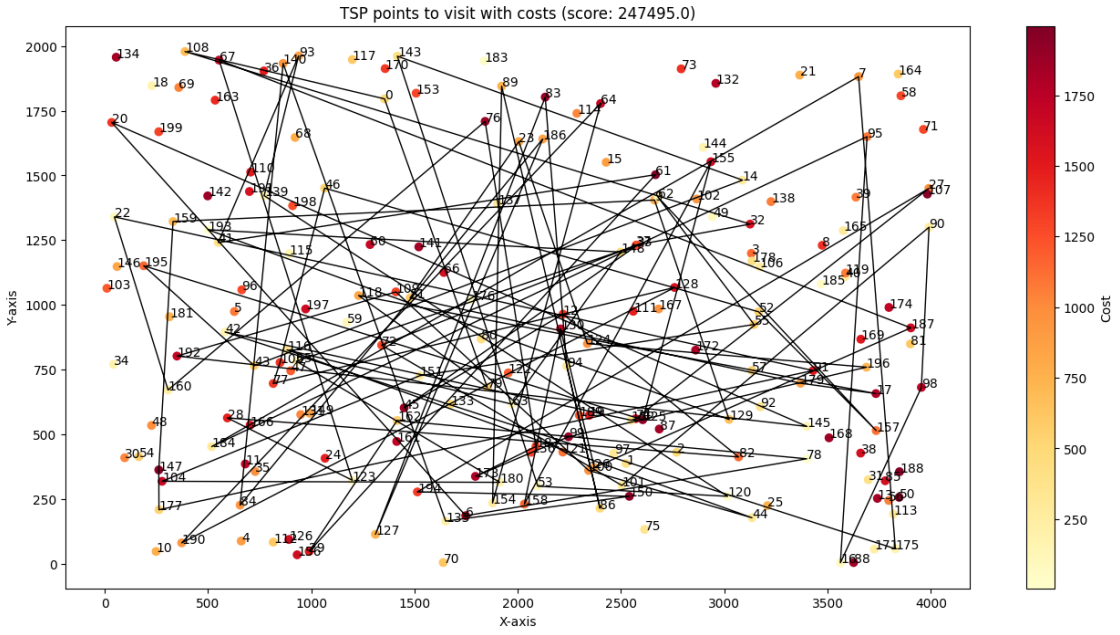
------- Solution end -------
---------- TSPA.csv NNHead
[0, 117, 46, 68, 93, 140, 36, 163, 199, 146, 195, 103, 5, 96, 118, 149, 131,
112, 4, 84, 35, 10, 190, 127, 70, 101, 97, 1, 152, 120, 78, 145, 185, 40, 165,
90, 81, 113, 175, 171, 16, 31, 44, 92, 57, 106, 49, 144, 62, 14, 178, 52, 55,
129, 2, 75, 86, 26, 100, 121, 124, 94, 63, 53, 180, 154, 135, 123, 65, 116, 59,
115, 139, 193, 41, 42, 160, 34, 22, 18, 108, 69, 159, 181, 184, 177, 54, 30, 48,
43, 151, 176, 80, 79, 133, 162, 51, 137, 183, 143]



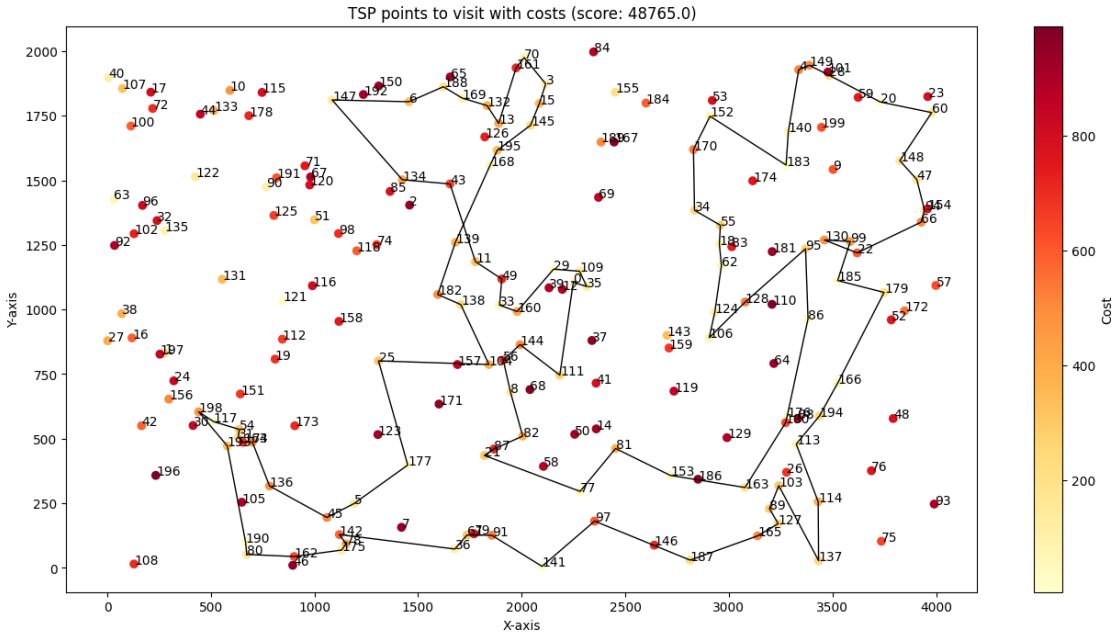
------- Solution end -------
---------- TSPA.csv NNWhole
[0, 143, 186, 23, 89, 183, 137, 176, 51, 118, 115, 68, 46, 139, 41, 108, 18,
146, 22, 159, 193, 34, 184, 190, 10, 177, 30, 54, 160, 181, 42, 43, 116, 131,
149, 65, 59, 127, 84, 4, 112, 123, 70, 135, 154, 180, 75, 86, 101, 90, 165, 196,
81, 119, 40, 144, 148, 9, 62, 14, 102, 49, 178, 55, 52, 106, 185, 57, 129, 92,
179, 145, 31, 56, 113, 175, 171, 16, 78, 25, 44, 120, 2, 1, 152, 100, 26, 97,
53, 124, 94, 79, 63, 133, 162, 151, 80, 140, 93, 117]



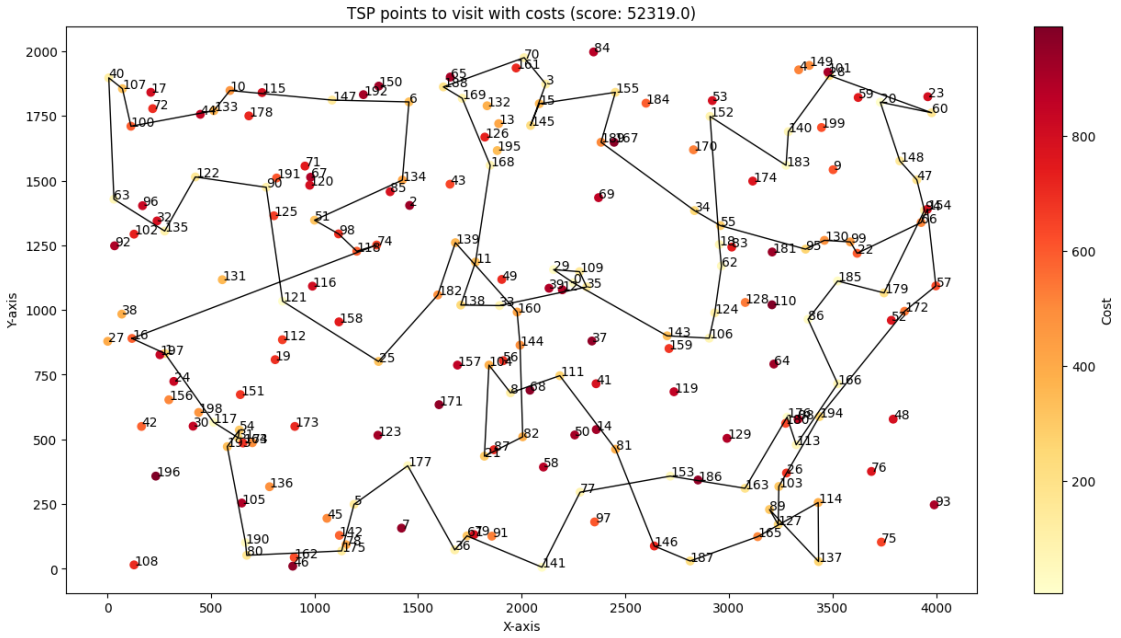

```
------- Solution end -------
---------- TSPA.csv RandomSolver
[0, 79, 118, 17, 179, 51, 23, 124, 196, 99, 173, 182, 76, 160, 22, 91, 177, 147,
159, 9, 129, 193, 148, 63, 116, 44, 100, 109, 192, 145, 20, 6, 78, 162, 29, 61,
41, 93, 139, 120, 53, 143, 14, 43, 195, 125, 155, 158, 130, 26, 187, 67, 123,
28, 82, 12, 45, 33, 65, 180, 104, 151, 55, 62, 157, 52, 46, 105, 77, 128, 190,
64, 154, 89, 86, 94, 83, 127, 186, 84, 140, 135, 150, 194, 27, 74, 152, 42, 175,
7, 184, 95, 16, 98, 90, 101, 72, 37, 32, 108]
```



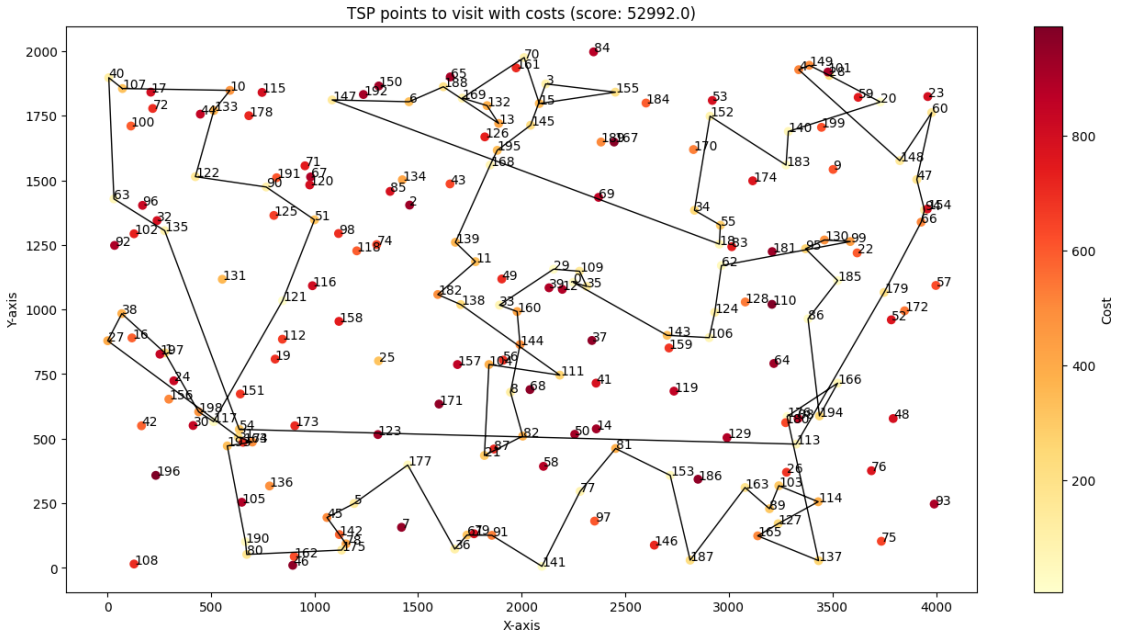
------- Solution end -------
---------- TSPB.csv GreedyCycle
[0, 35, 109, 29, 160, 33, 49, 11, 43, 134, 147, 6, 188, 169, 132, 13, 161, 70,
3, 15, 145, 195, 168, 139, 182, 138, 104, 25, 177, 5, 45, 136, 73, 164, 31, 54,
117, 198, 193, 190, 80, 162, 175, 78, 142, 36, 61, 91, 141, 97, 187, 165, 127,
89, 103, 137, 114, 113, 194, 166, 179, 185, 99, 130, 22, 66, 94, 47, 148, 60,
20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 86, 176,
180, 163, 153, 81, 77, 21, 87, 82, 8, 56, 144, 111]



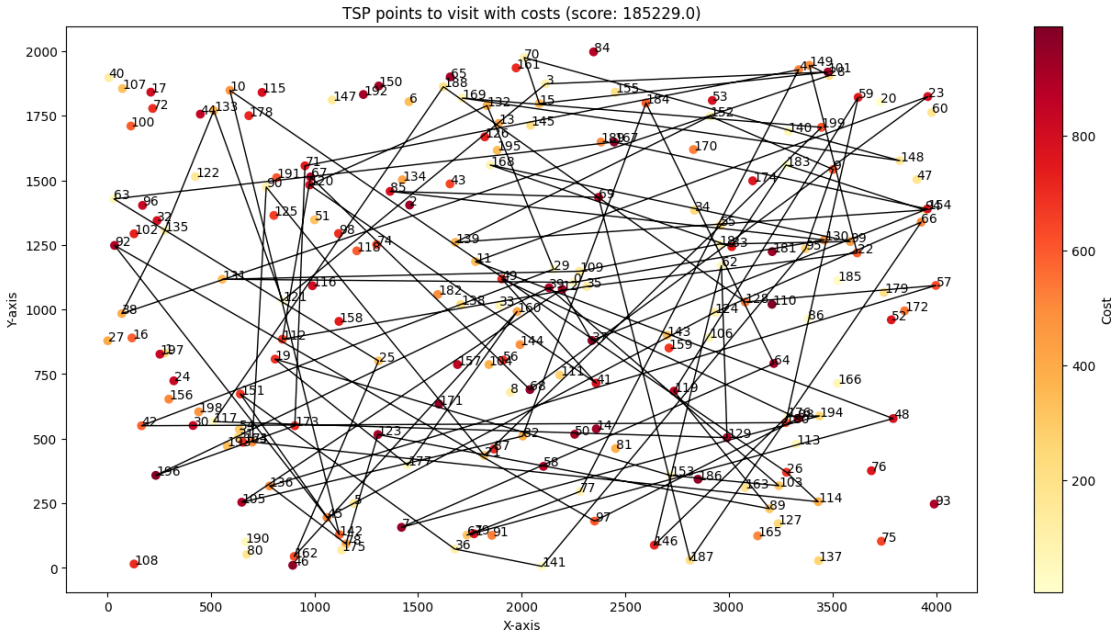
------- Solution end -------
---------- TSPB.csv NNHead
[0, 29, 109, 35, 33, 138, 11, 168, 169, 188, 70, 3, 145, 15, 155, 189, 34, 55,
95, 130, 99, 22, 66, 154, 57, 172, 194, 103, 127, 89, 137, 114, 165, 187, 146,
81, 111, 8, 104, 21, 82, 144, 160, 139, 182, 25, 121, 90, 122, 135, 63, 40, 107,
100, 133, 10, 147, 6, 134, 51, 98, 118, 74, 16, 1, 117, 31, 54, 193, 190, 80,
175, 5, 177, 36, 61, 141, 77, 153, 163, 176, 113, 166, 86, 185, 179, 94, 47,
148, 20, 60, 28, 140, 183, 152, 18, 62, 124, 106, 143]



------- Solution end -------
---------- TSPB.csv NNWhole
[0, 143, 106, 124, 62, 99, 130, 95, 185, 86, 194, 166, 176, 137, 165, 127, 114, 103, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 177, 5, 45, 142, 78, 175, 80, 190, 193, 73, 31, 27, 38, 1, 198, 117, 121, 51, 90, 122, 133, 10, 107, 40, 63, 135, 54, 113, 179, 66, 94, 47, 60, 148, 4, 149, 28, 20, 140, 183, 152, 34, 55, 18, 147, 6, 188, 13, 132, 169, 70, 15, 155, 3, 145, 195, 168, 139, 11, 182, 138, 111, 104, 21, 82, 8, 144, 160, 33, 29, 109, 35]



```
------- Solution end -------
---------- TSPB.csv RandomSolver
[0, 131, 109, 18, 99, 168, 83, 183, 94, 139, 152, 9, 95, 141, 36, 151, 142, 10,
26, 114, 73, 90, 21, 199, 169, 148, 70, 15, 101, 3, 121, 133, 135, 38, 28, 149,
22, 35, 112, 67, 68, 23, 11, 48, 61, 77, 4, 71, 173, 196, 39, 103, 153, 19, 177,
160, 162, 25, 63, 167, 154, 37, 12, 105, 82, 184, 64, 7, 194, 176, 146, 62, 129,
117, 188, 128, 57, 136, 78, 123, 89, 119, 59, 180, 42, 29, 132, 120, 41, 49, 88,
97, 92, 45, 124, 66, 187, 69, 85, 130]
```




```
------- Solution end -------
```

1.6 Information whether the best solutions have been checked with the solution checker.

Yes solutions have been checked with solution checker

1.7 (Link to) the source code

https://github.com/BbqGamer/tsp

1.8 Conclusions

Greedy methods are nice in the sense that they are easy to implement and to run, in particular NNHead method was very fast (it works in linear time), even if the results weren't too good they were significantly better than random solution, however the solutions from NNHead result in a lot

of big jumps which is quite unoptimal. NNWhole results in slightly better results, but probably not enough to justify its higher complexity $O(n^2)$. The best Greedy algorithm out of three tested was GreedyCycle, it resulted in solutions that were more compressed, you can see a step improvement in the results however for the price of the complexity, this algorithm works in $O(n^3)$