Evolutionary Computation - Assignment 1

Sebastian Chwilczyński 148248, Karol Cyganik 148250

October 15, 2023

1 Description of the problem

We are given a graph with n n nodes, each of which is described by it's x and y coordinates and a node cost. The goal is to select exactly $ceil(n/2)$ nodes that form a Hamiltonian cycle (closed path) and minimize

$$\sum_{e \in E} cost(e) + \sum_{v \in V} cost(v)$$

Where E is a set of selected edges, $cost(e)$ is Euclidean distance between two nodes rounded mathematically to integer value, V is set of selected nodes and $cost(v)$ is node cost.

2 Pseudocode

2.1 Random Solution

Algorithm 1 Algorithm that randomly selects $ceil(n/2)$ nodes assuming that every pair of nodes is connected by an arc

Input: number of nodes n
Output: list of $ceil(n/2)$ random numbers

1: $cycle \leftarrow []$
2: **while** $len(cycle) < ceil(n/2)$ **do**
3: $\quad candidate \leftarrow$ random integer in range [0,n)
4: $\quad$ **if** *candidate* is not in *cycle* **then**
5: $\quad\quad$ add *candidate* to the *cycle*
6: $\quad$ **end if**
7: **end while**
8: **return** *cycle*

2.2 Nearest Neighbour

Algorithm 2 Algorithm that starts from given node and iteratively adds edge and node with lowest cost

Input: D: Distance matrix; C: List of costs; n: Starting node;
Output: $Cycle$: List of greedily selected nodes

1: $cycle \leftarrow [n]$
2: **for** $i = 1$ to $ceil(n/2)$ **do**
3: $\quad node \leftarrow \underset{n \notin cycle}{\arg\min}\ D[cycle[i], n] + C[n]$
4: $\quad$ append *node* to the *cycle*
5: **end for**
6: **return** *cycle*

2.3 Greedy Cycle

Algorithm 3 Algorithm that starts from given node and iteratively adds edge and node with lowest cost always creating a cycle

Input: D: Distance matrix; C: List of costs; n: Starting node;
Output: $Cycle$: List of greedily selected nodes

1: $chaepest \leftarrow \arg\min_{i \neq n} D[n, i] + C[i]$
2: $cycle \leftarrow [n, cheapest]$
3: **for** $i = 1$ to $ceil(n/2)$ **do**
4: $\quad position,\ vertex \leftarrow \arg\min_{j=0,1,\ldots,i, s \notin cycle} D[cycle[j], s] + D[s, cycle[j+1]] - D[cycle[j], cycle[j+1]] + C[s]$
5: $\quad$ insert $vertex$ at position $j+1$ to the $cycle$
6: **end for**
7: **return** $cycle$

3 Results

	random			nn		
Problem	Best	Worst	Average	Best	Worst	Average
TSPA	238331.0	291490.0	265228.75	84471.0	90513.0	87679.135
TSPB	283627.0	292368.0	266623.445	77448.0	82631.0	79282.58
TSPC	193432.0	238238.0	215074.295	56304.0	63697.0	58872.68
TSPD	192364.0	241940.0	218404.675	50335.0	59846.0	54290.68

	greedy		
Problem	Best	Worst	Average
TSPA	75666.0	80321.0	77064.415
TSPB	68743.0	76324.0	70735.35
TSPC	53226.0	58876.0	55842.07
TSPD	50409.0	60964.0	54883.01

Table 1: Comparison of TSP algorithms

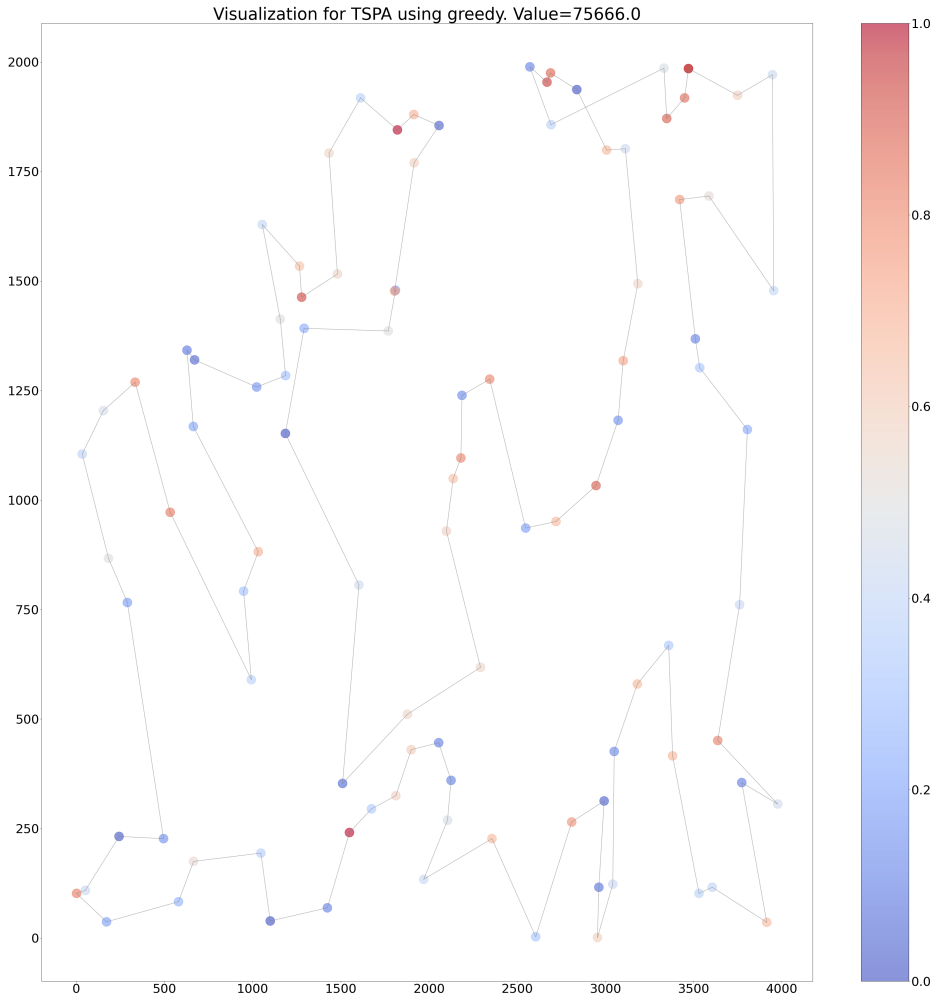

Figure 1: TSPA: Greedy

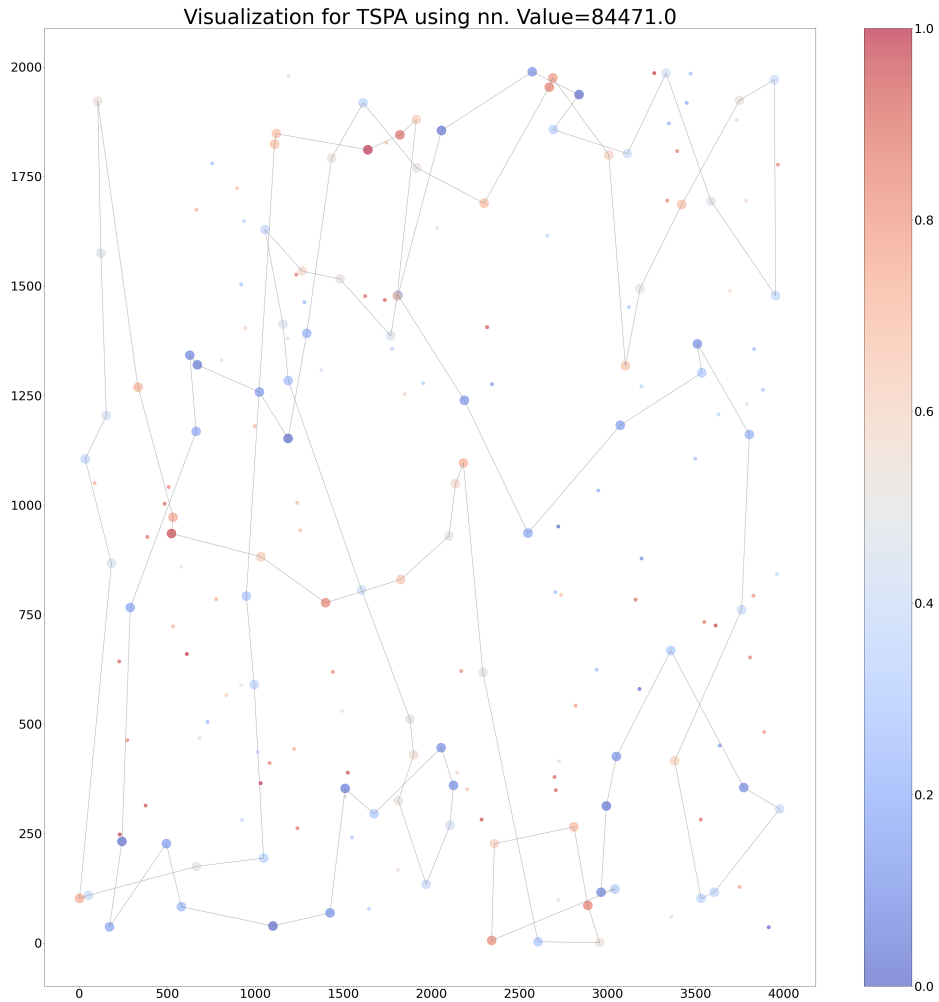

Figure 2: TSPA: Nearest Neighbor

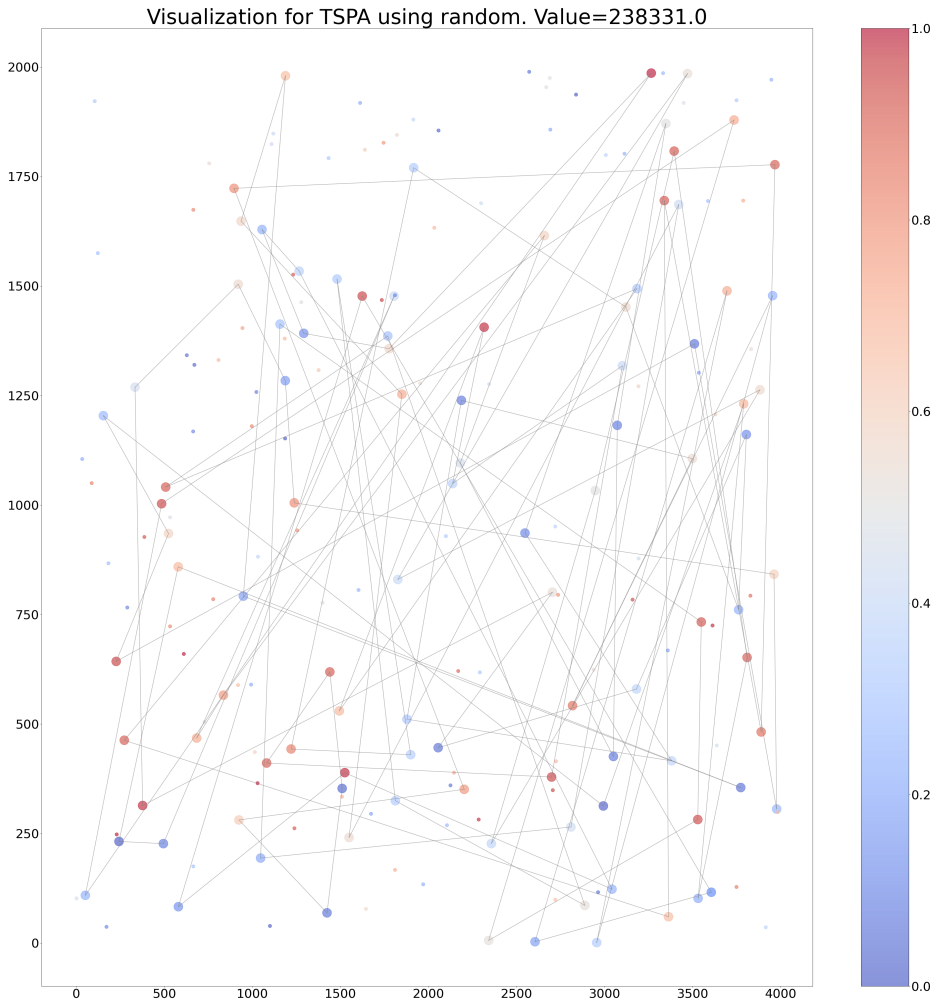

Figure 3: TSPA: Random

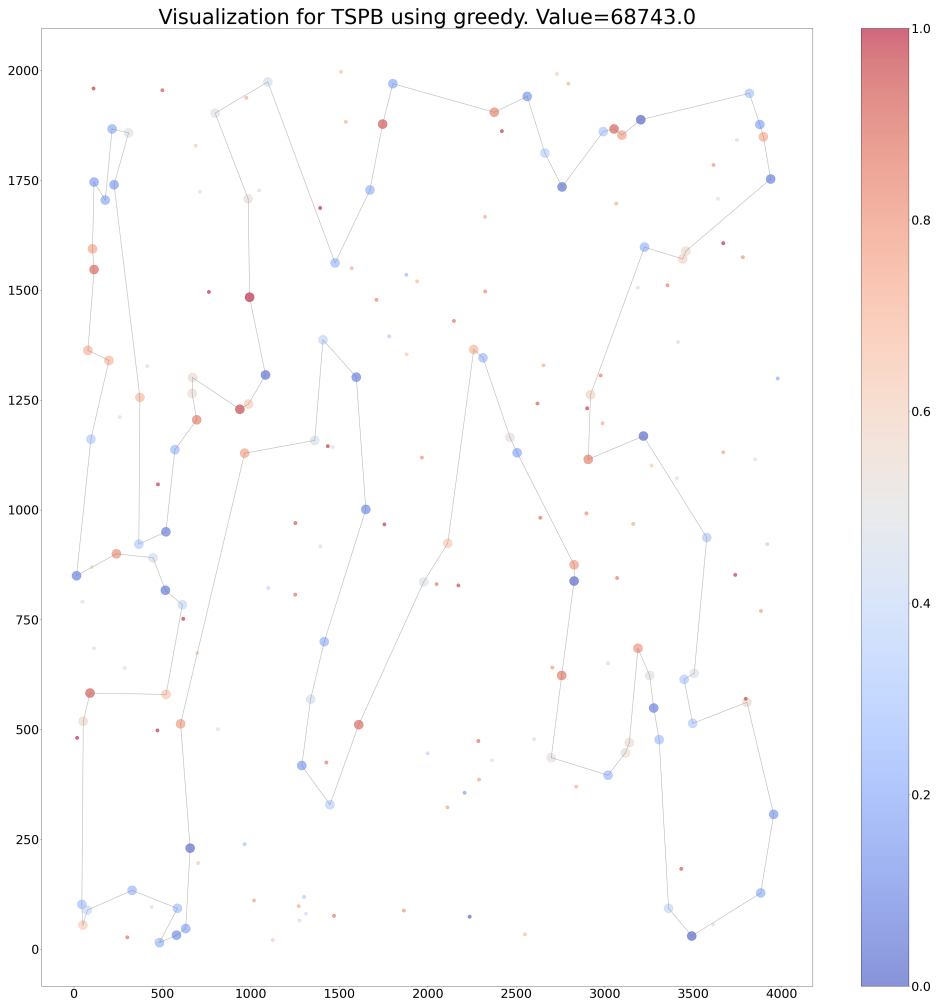

Figure 4: TSPB: Greedy

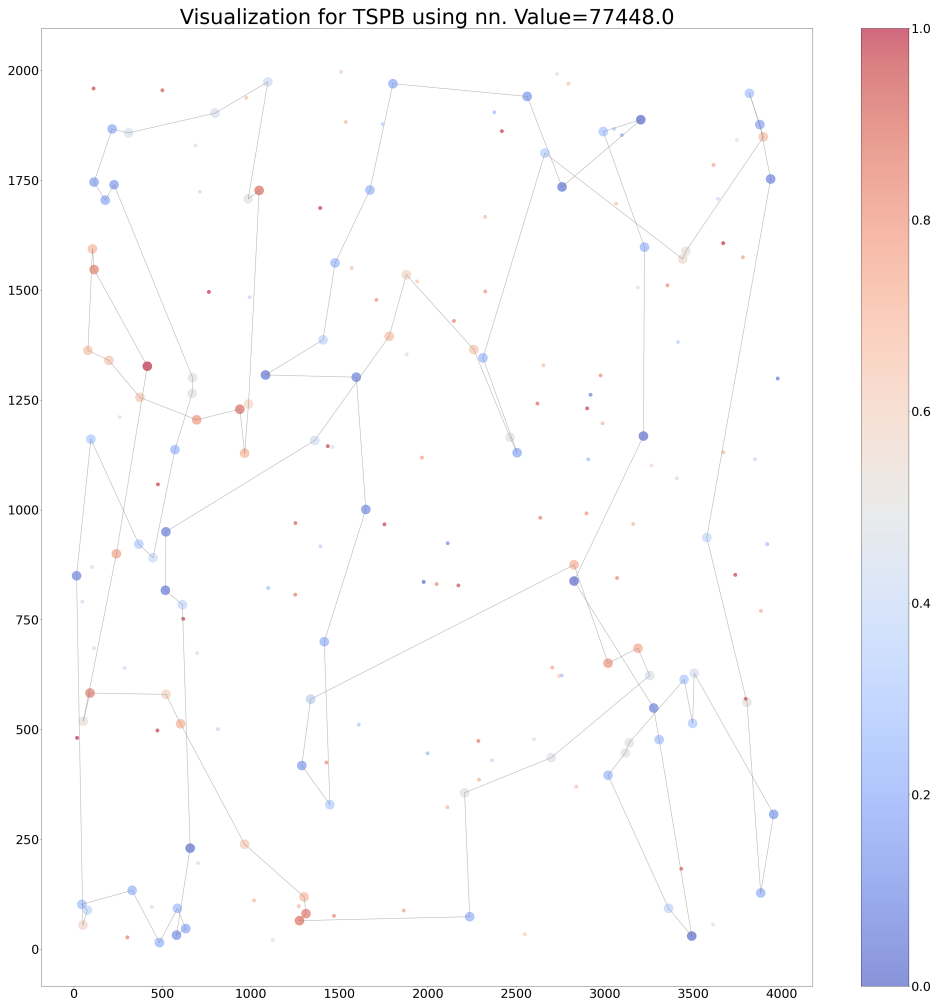

Figure 5: TSPB: Nearest Neighbor

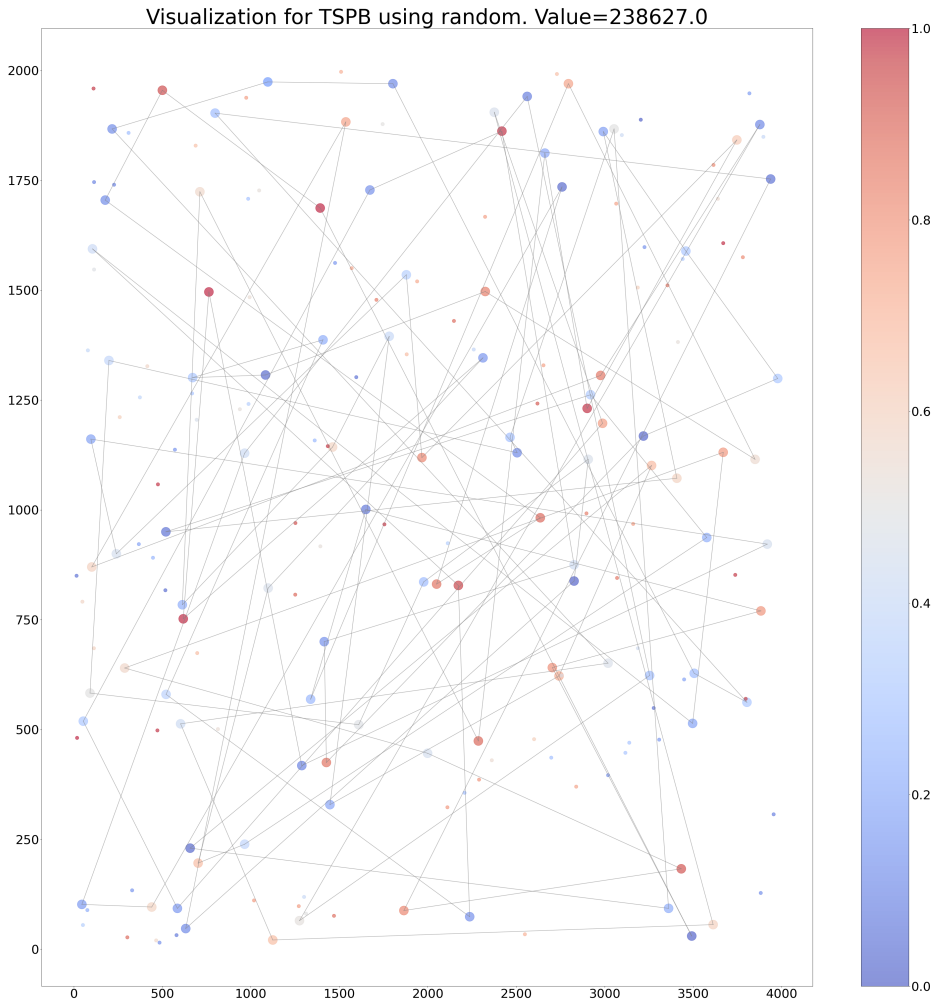

Figure 6: TSPB: Random

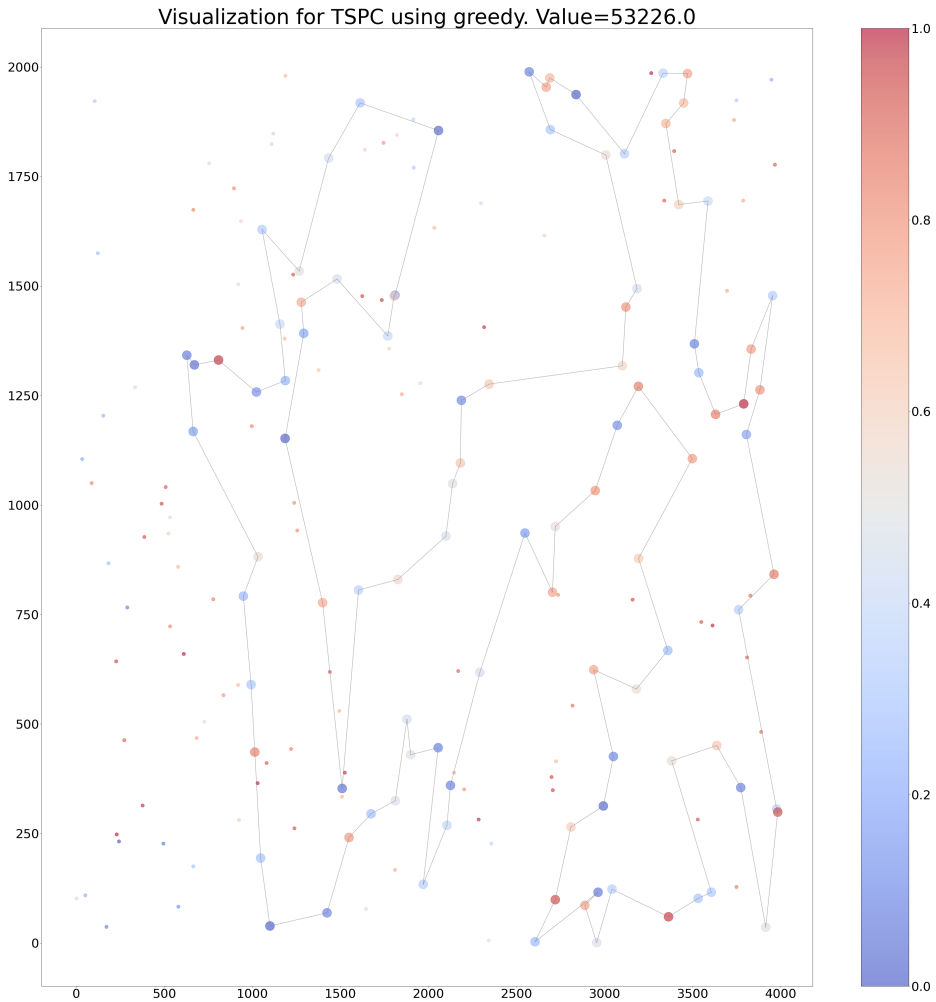

Figure 7: TSPC: Greedy

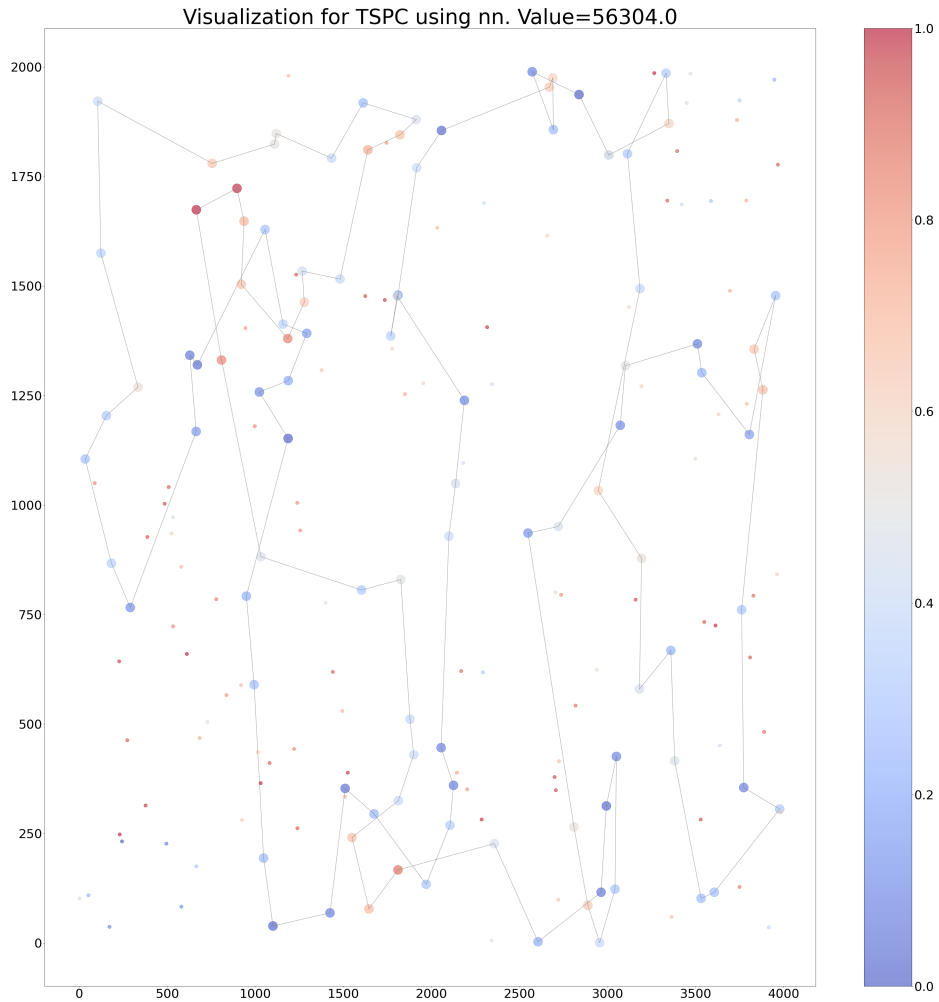

Figure 8: TSPC: Nearest Neighbor

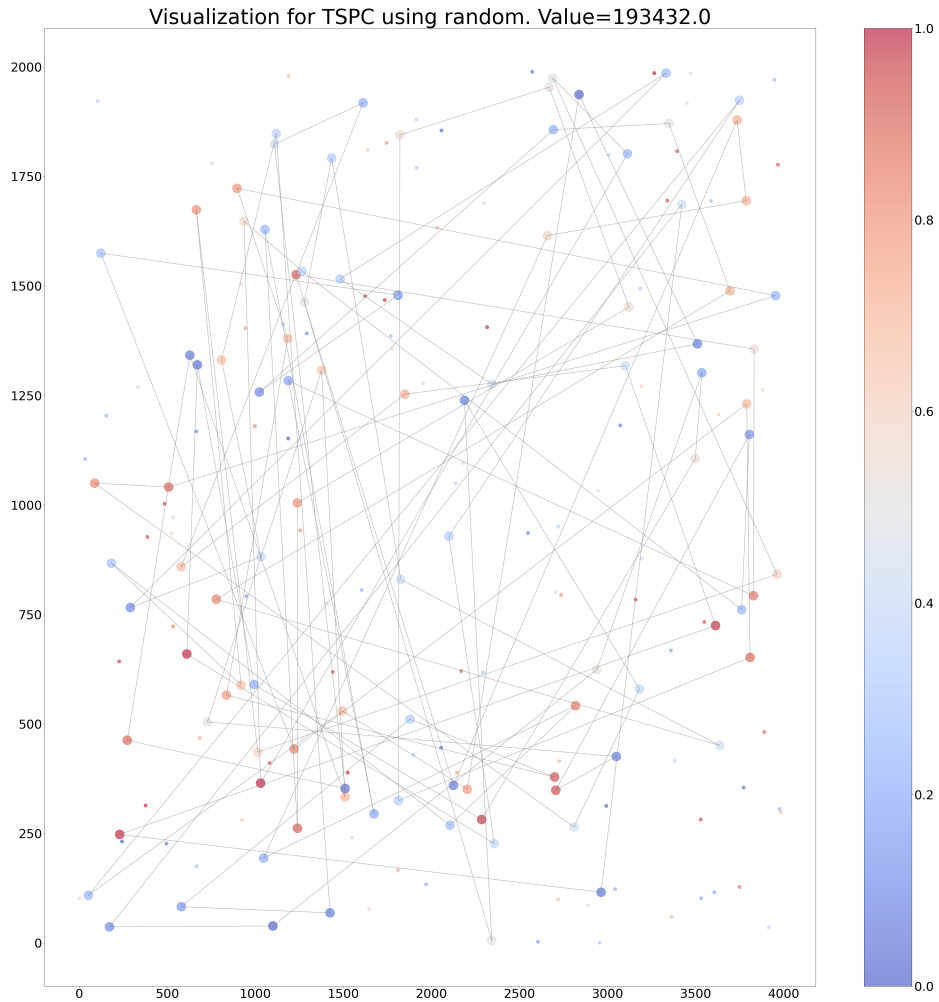

Figure 9: TSPC: Random

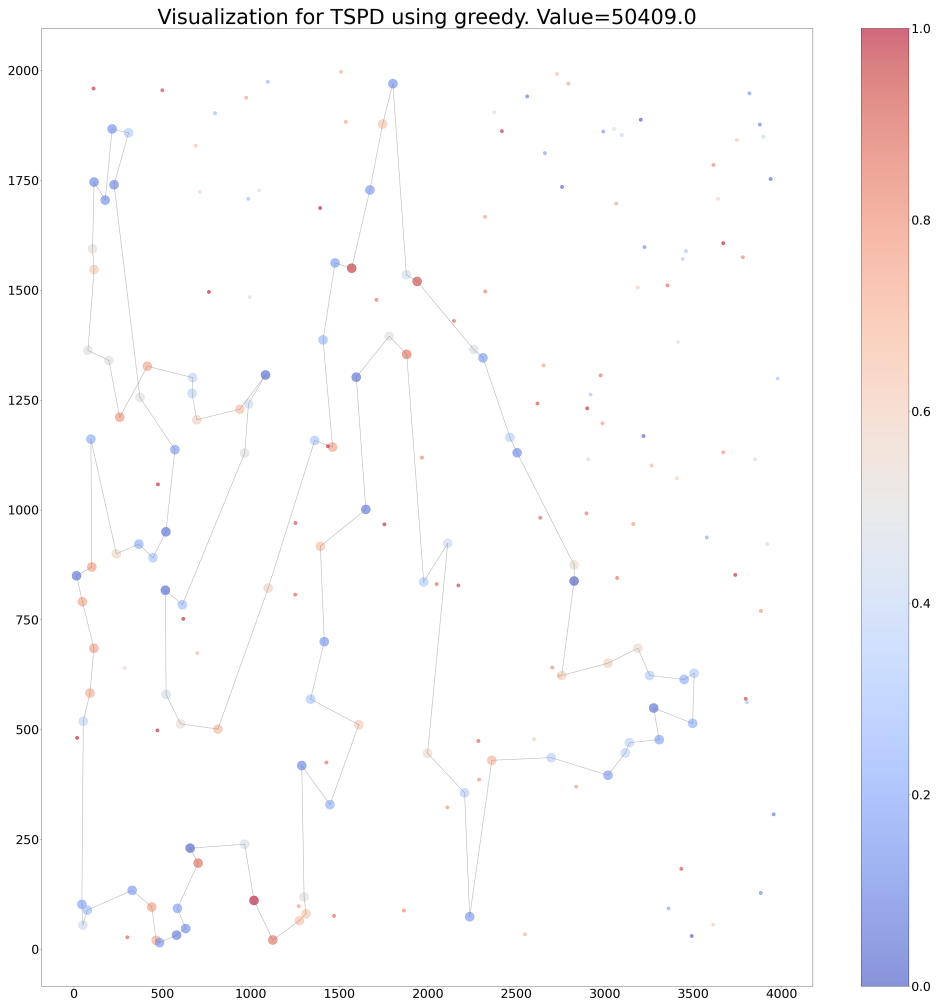

Figure 10: TSPD: Greedy

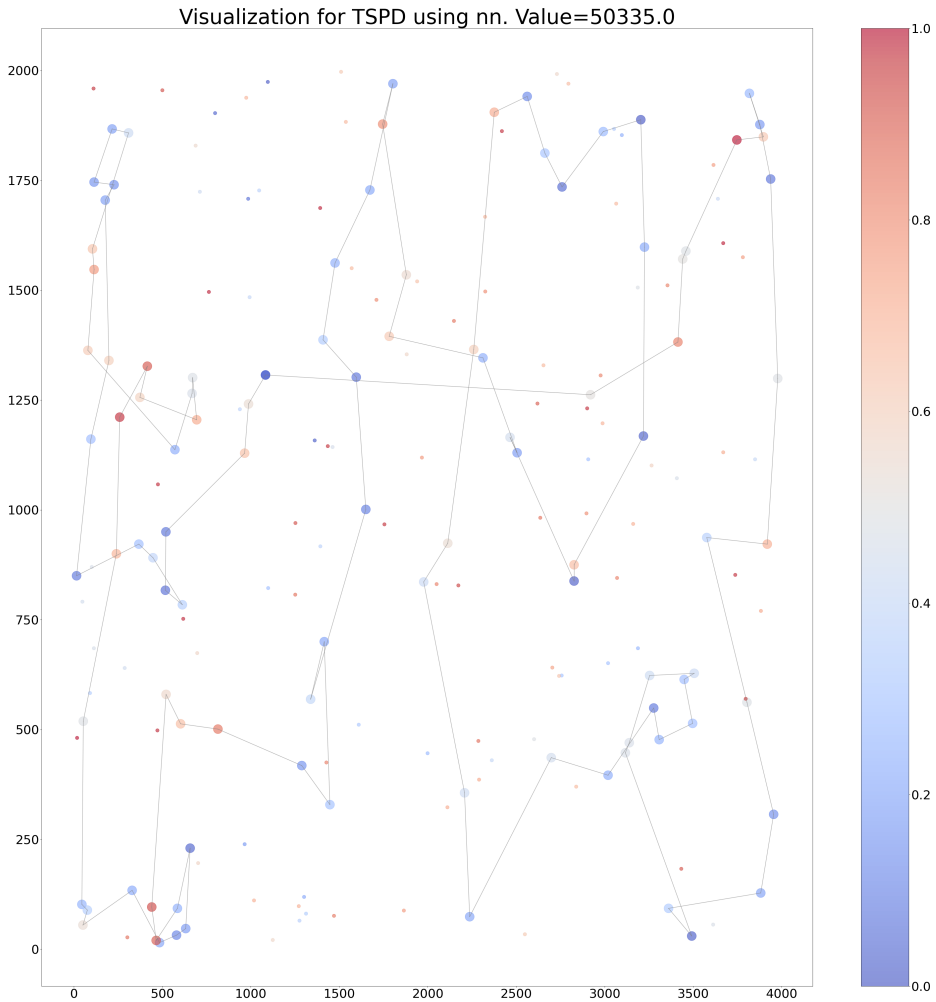

Figure 11: TSPD: Nearest Neighbor

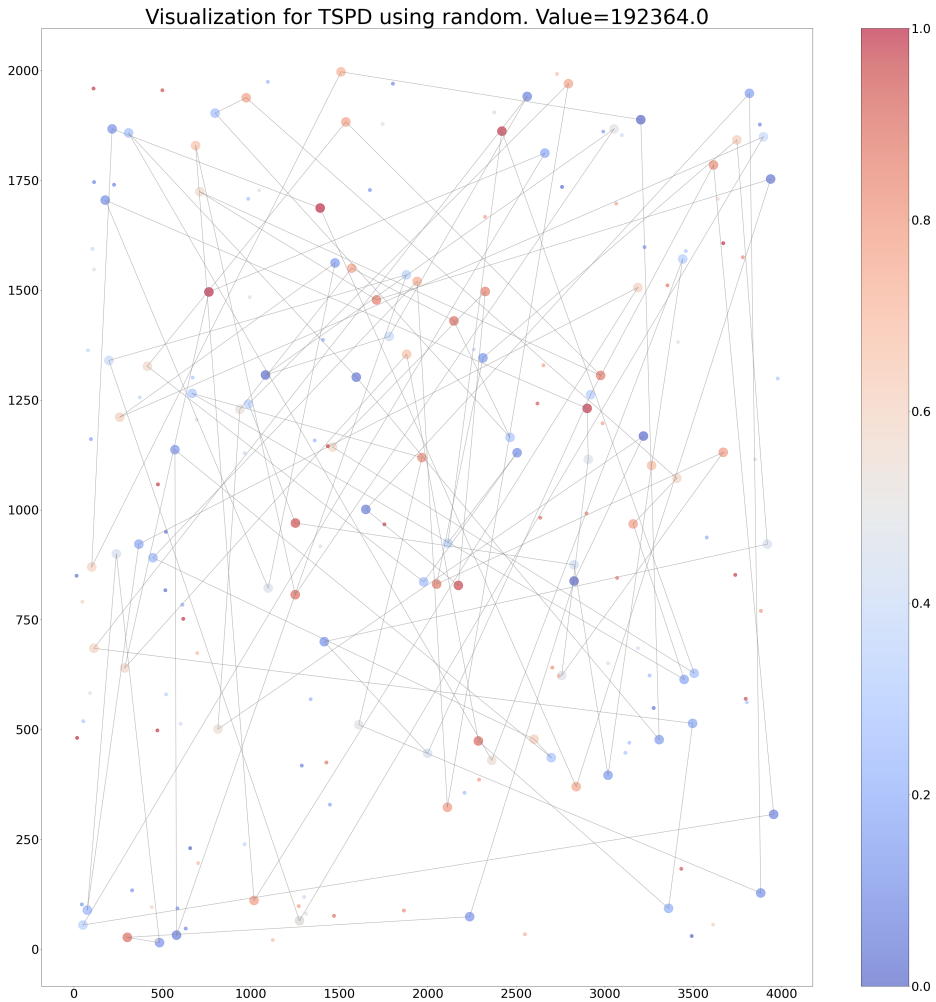

Figure 12: TSPD: Random

4 Code

Implementation of algorithms and visualizations is available here

5 Conclusions

In every problem, the greedy cycle method has the best results. On the visualizations we can see that paths created by the greedy cycle look mostly reasonable, the solution could be improved in several small areas, but generally, it is the best among others. The nearest neighbor method has slightly worse results, but the solutions often look strange when we look at them as humans. As expected, the random method ha the worst results with no intuitive paths and solutions far from being optimal.