Evolutionary Computation - lab assignment 1

Szymon Bujowski, 148050, source: https://github.com/bujowskis/put-evolutionary-computation

Problem description

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.

The task is to implement 4 methods, adapted to the problem:

- Random solution
- Nearest neighbor considering adding the node only at the end of the current path
- 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
- Greedy cycle

For each greedy method generate 200 solutions starting from each node. Generate also 200 random solutions.

Implemented algorithms and pseudocodes

Notes

- 50% was the required number of nodes in a solution, and “required number of nodes” will be referred to this way in the pseudocode
- “**Total move cost**” is the total change in objective function after adding a node, that is, its edge distance and additional cost

Random solution

```
1. Sample 50% of all nodes
2. Form the cycle from the chosen nodes
```

Nearest Neighbor - adding at end of path

```
1. Choose a starting node
2. Keep track of chosen and remaining nodes. Chosen nodes form the path
3. WHILE number of chosen nodes is not 50% AND there are still remaining nodes
    3.1. Determine total move cost from the last chosen node to all remaining nodes
    3.2. Choose a node with the smallest total move cost
    3.3. Append the node at end of chosen nodes, remove it from remaining nodes
4. Form the cycle from the chosen nodes
```

Nearest Neighbor - adding at any place (end, beginning, or inside current path)

```
1. Choose a starting node
2. Keep track of chosen and remaining nodes. Chosen nodes form the path
3. WHILE number of chosen nodes is not 50% AND there are still remaining nodes
    3.1. Determine total move cost from the last chosen node to all remaining nodes (case A)
    3.2. Determine total move cost from the first chosen node to all remaining nodes (case B)
    3.3. Determine total move costs of inserting any of remaining nodes between inner-path pairs of chosen nodes (case C)
    3.4. Choose a node with the smallest total move cost
    3.5. Add the node at its respective place in the path, depending on (case A,B,C), remove it from remaining nodes
        3.5.A. IF (case A) THEN append the node at end of path
        3.5.B. IF (case B) THEN add node at the beginning of path
        3.5.C. IF (case C) THEN insert node between its respective pair
4. Form the cycle from the chosen nodes
```

Greedy cycle

```
1. Choose a starting node
2. Keep track of chosen and remaining nodes. Chosen nodes form the cycle
    2.Ad.1. 1 node is treated as cycle to itself
    2.Ad.2. 2 nodes are treated as cycle back and forth
3. WHILE number of chosen nodes is not 50% AND there are still remaining nodes
3.1. Determine total move costs of inserting any of remaining nodes between pairs of chosen nodes
    3.1.Ad.1. Including nodes at start and end of the path forming the cycle
3.2. Choose a node with the smallest total move cost
3.3. Insert the node between its respective pair, remove it from remaining nodes
4. Form the cycle from the chosen nodes
```

Results

Note: All best solutions were checked with the solution checker

Statistics

TSPA

Statistic	Random	NN at end	NN at any	Greedy cycle
Min obj fun	236147	83182	71179	71488
Max obj fun	290929	89433	75450	74350
Avg obj fun	265351.24	85108.51	73172.74	72589.82
Min time (s)	0.00001	0.00100	0.14759	0.14422
Max time (s)	0.00133	0.00366	0.19540	0.15667
Avg time (s)	0.00008	0.00237	0.15767	0.14755

TSPB

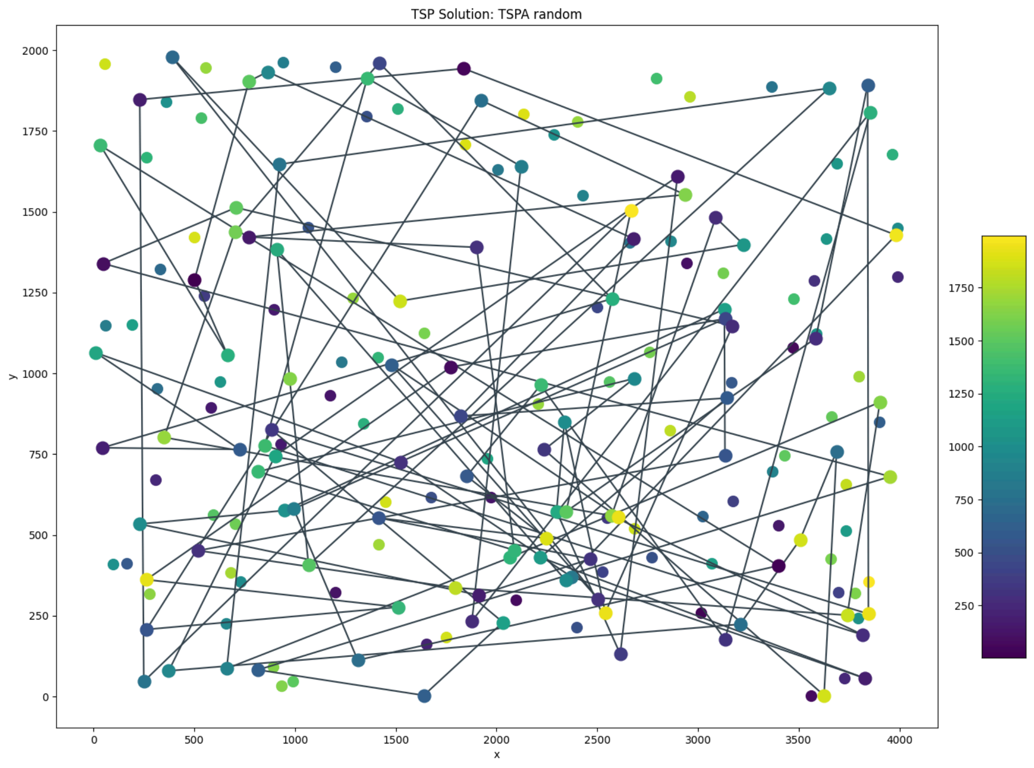
Statistic	Random	NN at end	NN at any	Greedy cycle
Min obj fun	193230	52319	44417	48765
Max obj fun	236483	59030	53438	57262
Avg obj fun	213807.54	54390.43	45870.26	51389.38
Min time (s)	0.00001	0.00146	0.15079	0.14758
Max time (s)	0.00150	0.00376	0.53195	0.30969
Avg time (s)	0.00008	0.00241	0.16658	0.15973

Best solutions

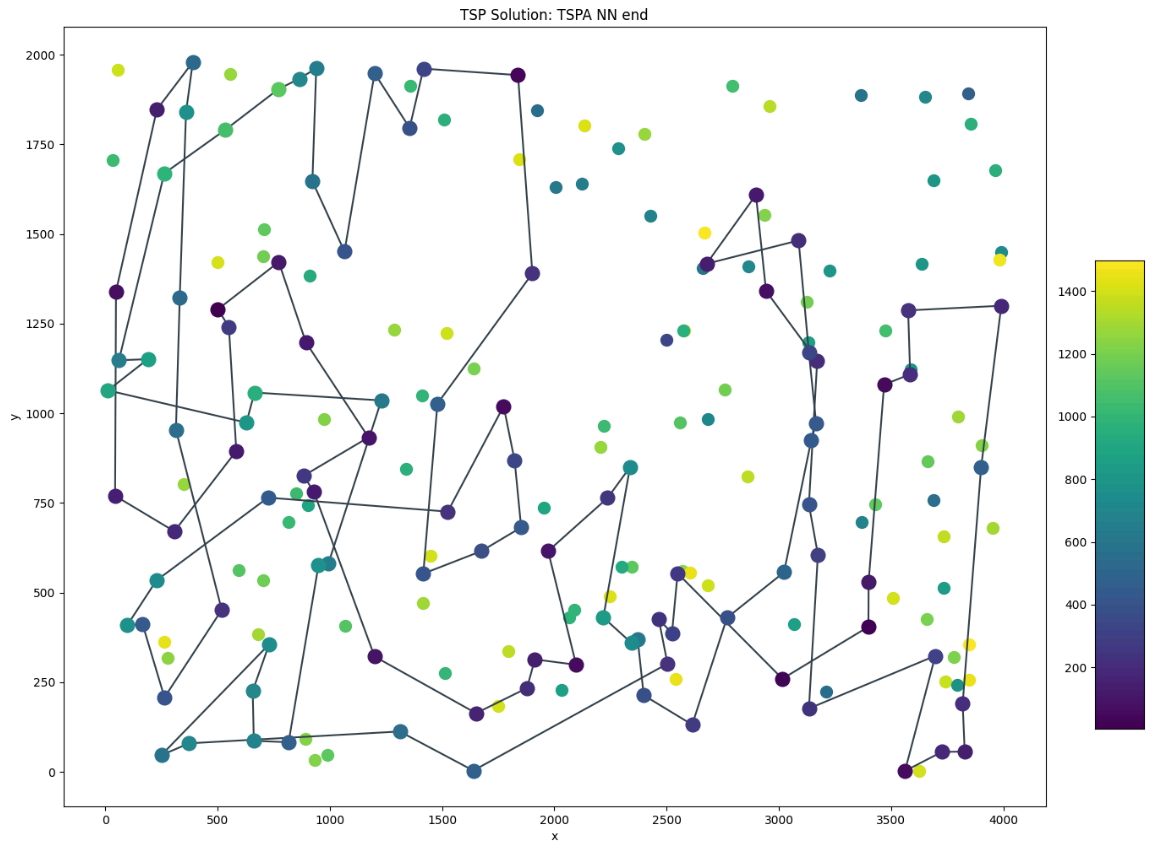
Note: additional cost is depicted using a color scale

TSPA

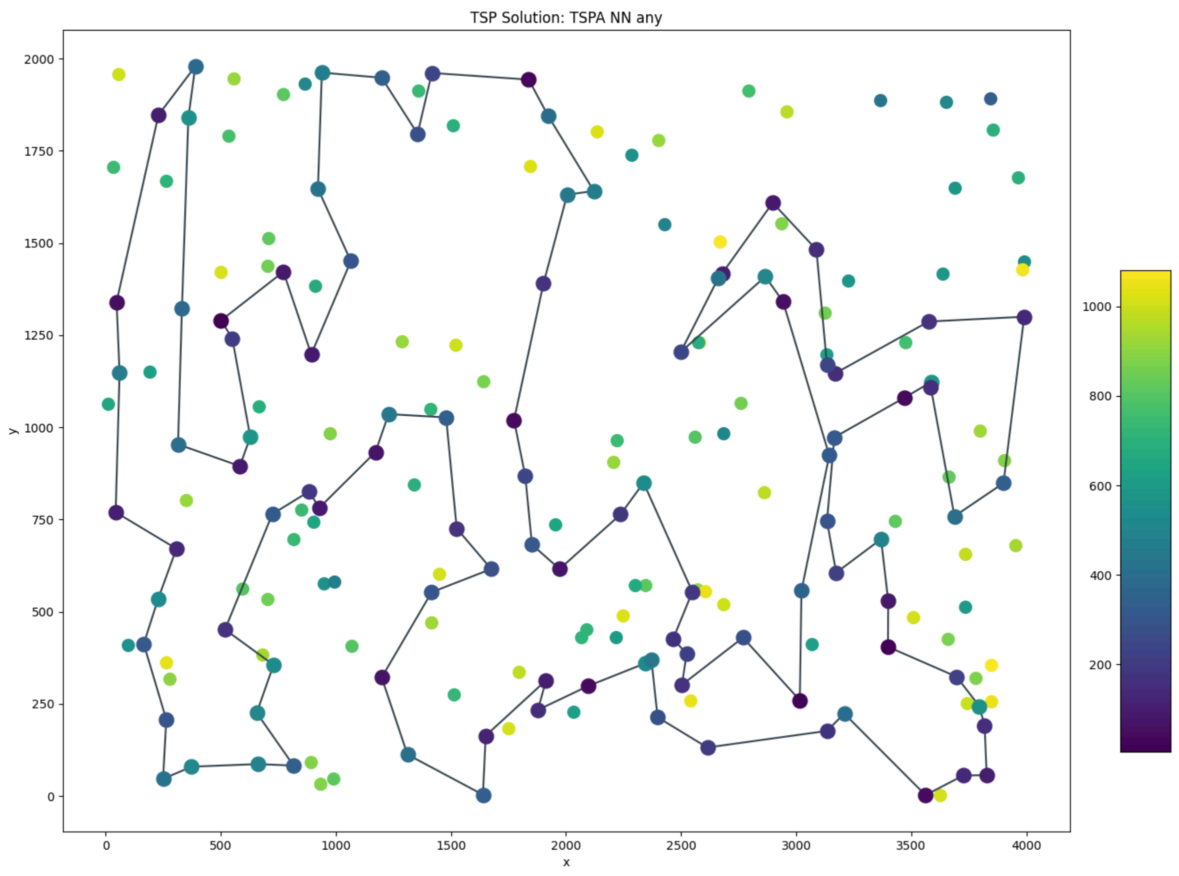
Random solution - random-best.json



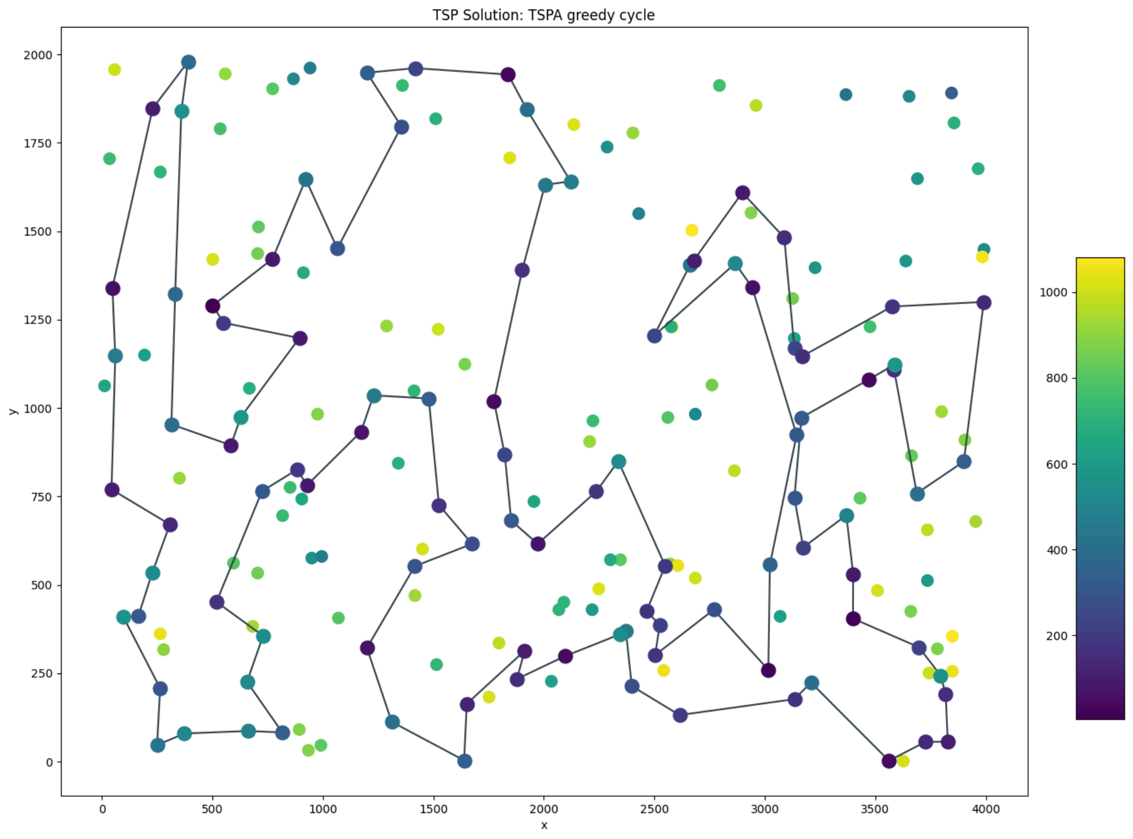
NN at end - nn-end-best.json



NN at any - [nn-any-best.json]()

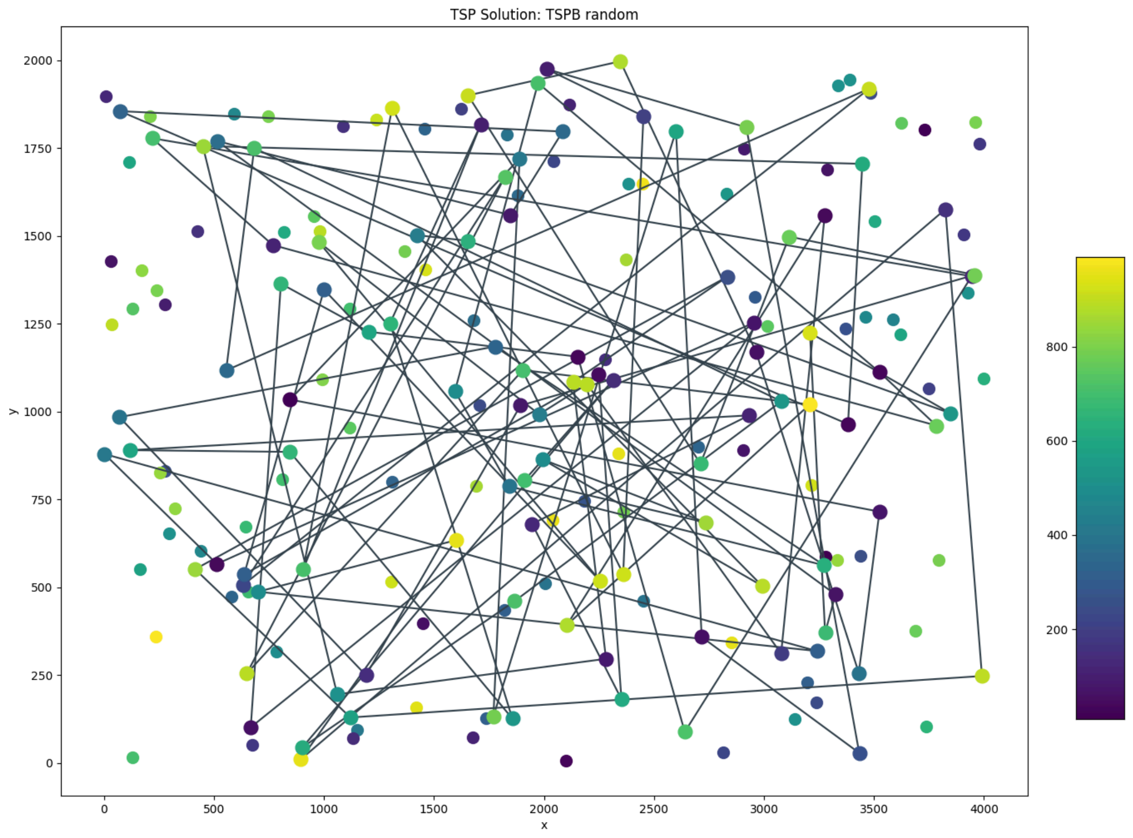


Greedy cycle - greedy-cycle-best.json

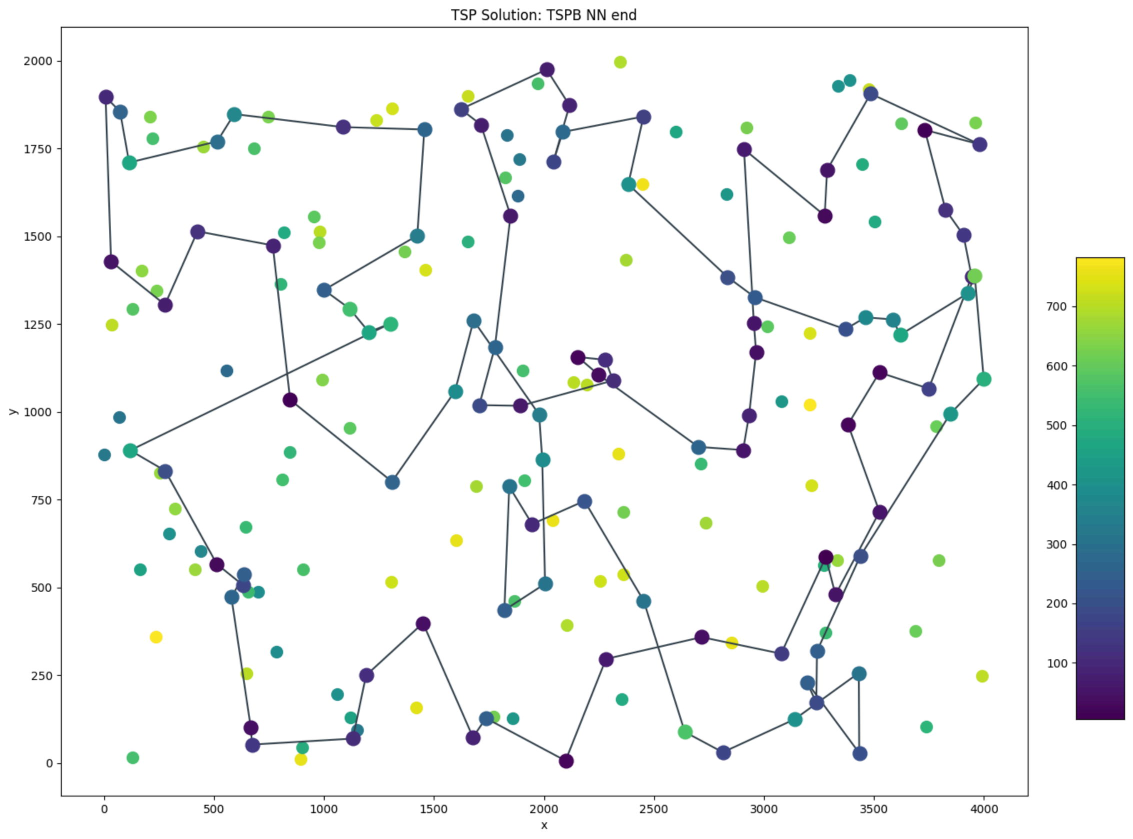


TSPB

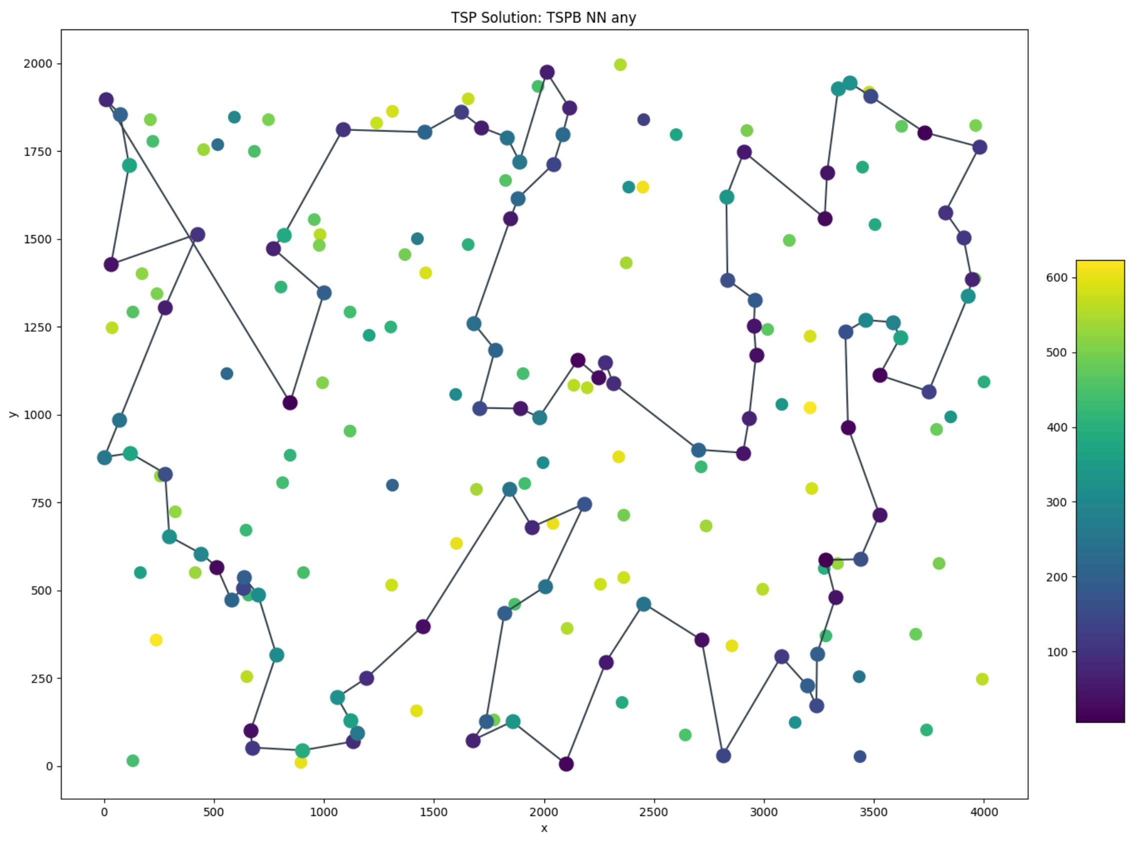
Random solution - random-best.json



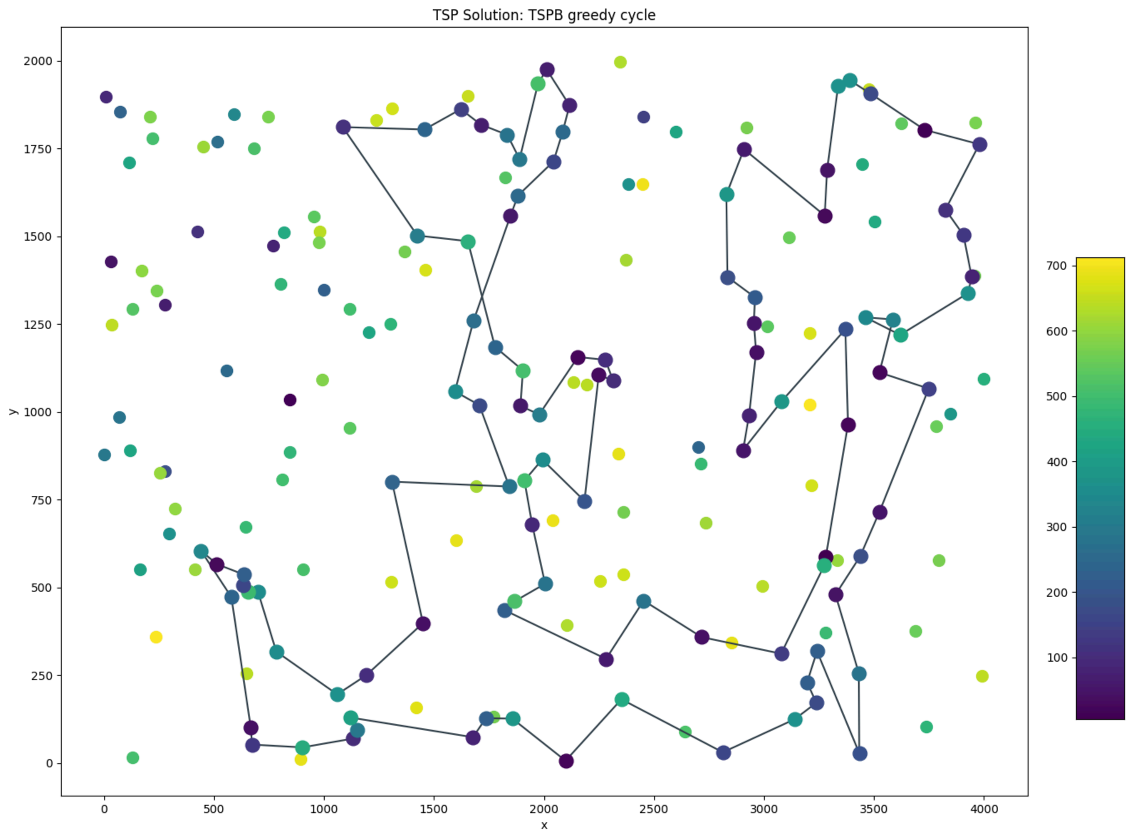
NN at end - [nn-end-best.json]



NN at any - [nn-any-best.json]



Greedy cycle - greedy-cycle-best.json



Conclusions

- Random, while extremely light computationally, is insufficient to create good solutions - even the most basic heuristic (NN at end) outperforms it by at least around factor of 3
- Both NN and greedy cycle heuristics can consistently create relatively good solutions
- NN at end offers simplicity and much faster computation than NN at any or greedy cycle - if coupled with not as big need for quality of the solution, it may be viable for some applications
- NN at any at least slightly outperforms greedy cycle, with similar computational demand - its ability to simply extend the path from start or end seem to help it strike a balance