Evolutionary Computation - lab assignment 3

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

Problem description

Implement local search. Local search should use deltas (changes) of the objective function. Do not use trivial implementation with neighbor solution constructed explicitly and the objective function calculated from the scratch. On the other hand, do not use more advanced techniques like deltas from previous iterations or candidate moves.

Type of local search: Implement both steepest and greedy version of local search. In greedy version the neighborhood should be browsed in random/randomized order. In the report please describe what kind of randomization was used.

Type of neighborhood: We will need two kinds of moves: intra-route moves – moves changing the order of nodes within the same set of selected nodes, inter-route moves – moves changing the set of selected nodes. As inter-route move use exchange of two nodes – one selected one not selected. For intra-route moves use two options: two-nodes exchange, two-edges exchange.

As inter-route moves use always nodes-exchange with one selected one not selected.

Note that the whole neighborhood is composed of moves of two kinds inter- and intra-route moves. In the steepest version we should select the best move from the whole neighborhood composed of moves of two types. In the greedy version ideally we should browse moves of two kinds in a random order. If we use a simpler randomization, we cannot, for example, start always from moves of a given kind (e.g. always from intra-route moves).

Type of starting solutions: Use two options: Random starting solutions. The best greedy construction heuristic (including regret heuristics) from the previous assignments with a random starting node.

Summarizing we have 8 combinations based on three binary options: type of local search, type of intra-route moves, type of starting solutions.

Computational experiment: Run each of the eight methods 200 times. For random starting solutions use 200 randomly generated solutions. For greedy starting solutions use each of the 200 nodes as the starting node for the greedy heuristic.

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

Local Search

```
1. Generate initial solution (random or greedy cycle, according to type of starting solution config)
2. Initialize list of all possible moves - that is, those resulting in decrease of objective function (inter nodes + intra nodes/edges exchanges, according to intra-route moves config)
     2.Ad.1. In case of steepest LS, evaluate and add only improving moves
3. WHILE there are possible moves
3.1. Choose a move (yielding best improvement or chosen randomly, according to type of local search)
     3.1.Ad.1. Choosing at random is achieved by taking move at random index of a list with all possible moves
     3.1.Ad.2. In case of greedy LS - evaluate move; in case it yields no improvement, keep choosing randomly
3.2. Apply the move on the cycle
3.3. Update list of all possible improving moves
4. Return the cycle
```

Results

Note: All best solutions were checked with the solution checker
Note: Local Search config is provided in order:
(search_type, initial_solution, intra_moves)

Statistics

Method	**TSPA** avg (min - max)	**TSPB** avg (min - max)
Random	265351 (236147 - 290929)	213808 (193230 - 236483)
NN at end	85109 (83182 - 89433)	54390 (52319 - 59030)
NN at any	73173 (71179 - 75450)	45870 (44417 - 53438)
Greedy cycle	72590 (71488 - 74350)	51389 (48765 - 57262)
Greedy 2-regret	99830 (85699 - 107981)	64565 (60110 - 70419)
Greedy 2-regret weighted objective	74037 (72590 - 76005)	56039 (52645 - 61863)
Local Search (greedy, greedy, two_edges)	72019 (69885 - 74249)	45224 (44076 - 50848)
Local Search (greedy, greedy, two_nodes)	72983 (71179 - 74942)	45749 (44392 - 52452)
Local Search (greedy, random, two_edges)	74122 (71115 - 77250)	48563 (46049 - 52622)
Local Search (greedy, random, two_nodes)	101920 (89295 - 111291)	75897 (66089 - 88866)
Local Search (steepest, greedy, two_edges)	71998 (70110 - 74270)	45171 (44140 - 51146)
Local Search (steepest, greedy, two_nodes)	72965 (71179 - 74942)	45744 (44392 - 52452)
Local Search (steepest, random, two_edges)	74222 (71725 - 78624)	48662 (46269 - 52049)
Local Search (steepest, random, two_nodes)	103693 (92916 - 121129)	76644 (68125 - 87325)

Running times (s)

Method	**TSPA** avg (min - max)	**TSPB** avg (min - max)
Random	0.00008 (0 - 0.00133)	0.00008 (0 - 0.00150)
NN at end	0.00237 (0.001 - 0.00366)	0.00240 (0.00145 - 0.00376)
NN at any	0.15767 (0.14758 - 0.1954)	0.16658 (0.15078 - 0.53)
Greedy cycle	0.1476 (0.14421 - 0.15667)	0.16 (0.14757 - 0.31)
Greedy 2-regret	0.54 (0.42 - 1.25)	0.44 (0.42 - 0.48)
Greedy 2-regret weighted objective	0.44 (0.43 - 0.48)	0.44 (0.43 - 0.53)
Local Search (greedy, greedy, two_edges)	0.59 (0.23 - 1.20)	0.46 (0.23 - 1.16)
Local Search (greedy, greedy, two_nodes)	0.23 (0.21 - 0.37)	0.23 (0.20 - 0.37)
Local Search (greedy, random, two_edges)	6.80 (4.87 - 9.36)	5.08 (4.03 - 6.05)
Local Search (greedy, random, two_nodes)	0.88 (0.53 - 1.11)	0.81 (0.54 - 1.12)
Local Search (steepest, greedy, two_edges)	0.39 (0.27 - 0.67)	0.32 (0.26 - 0.59)
Local Search (steepest, greedy, two_nodes)	0.25 (0.23 - 0.27)	0.25 (0.24 - 0.26)
Local Search (steepest, random, two_edges)	1.68 (1.35 - 2.19)	1.77 (1.34 - 2.14)
Local Search (steepest, random, two_nodes)	0.12 (0.08 - 0.30)	0.12 (0.10 - 0.19)

Local Search statistics

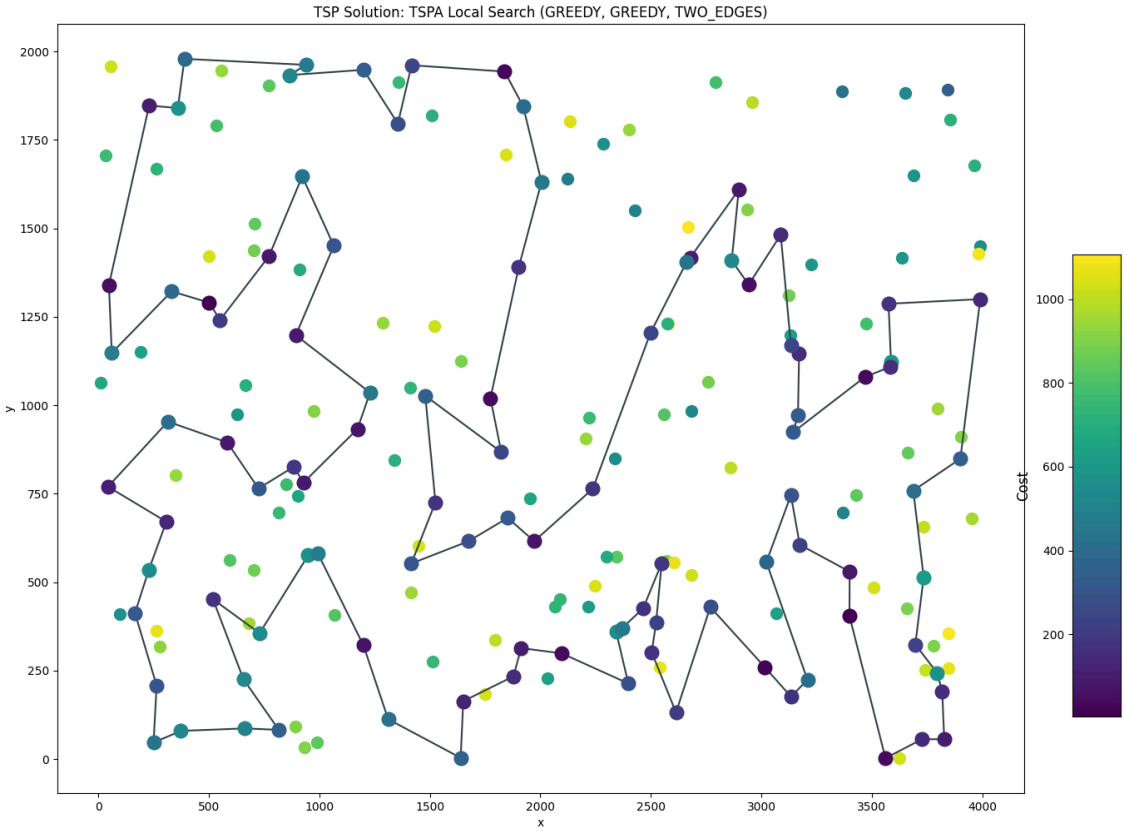
Note: additional statistics I thought may be interesting to investigate

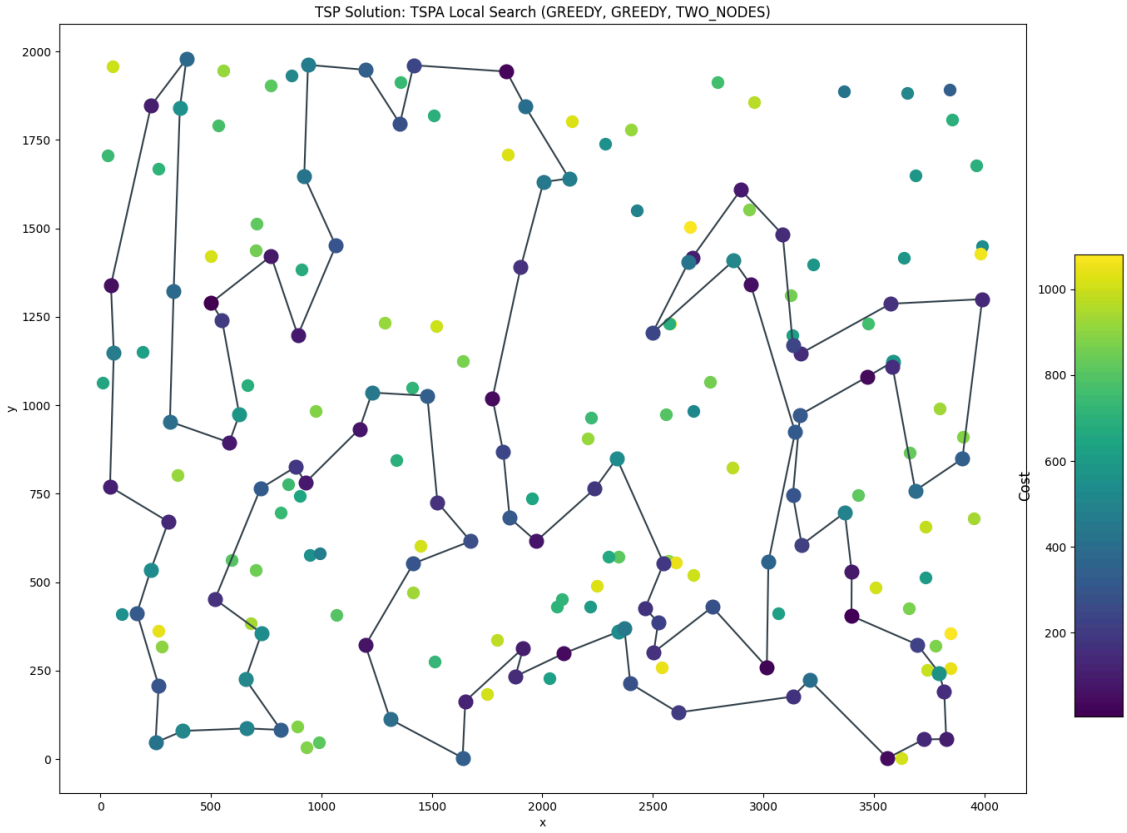
Method	**Moves evaluated** TSPA avg (min - max) TSPB avg (min - max)	**Total moves** TSPA avg (min - max) TSPB avg (min - max)	**Inter moves %** TSPA avg (min - max) TSPB avg (min - max)	**Intra moves %** TSPA avg (min - max) TSPB avg (min - max)
Local Search (greedy, greedy, two_edges)	48559 (14988 - 100819) 38185 (15450 - 104860)	11 (2 - 34) 8 (1 - 36)	13 (0 - 67) 28 (0 - 75)	87 (33 - 100) 72 (25 - 1)
Local Search (greedy, greedy, two_nodes)	10001 (9525 - 10563) 9999 (9370 - 10517)	2 (0 - 8) 2 (0 - 19)	85 (33 - 75) 93 (17 - 79)	15 (0 - 67) 7 (0 - 83)
Local Search (greedy, random, two_edges)	93558 (59752 - 148446) 94068 (60125 - 178500)	418 (370 - 481) 415 (351 - 486)	40 (33 - 47) 48 (39 - 56)	60 (53 - 67) 52 (44 - 61)
Local Search (greedy, random, two_nodes)	18456 (11087 - 29308) 17713 (11387 - 29785)	362 (241 - 443) 357 (278 - 490)	43 (33 - 52) 50 (33 - 59))	56 (48 - 67) 50 (41 - 67)
Local Search (steepest, greedy, two_edges)	118233 (44850 - 284647) 70026 (29900 - 255641)	8 (2 - 20) 5 (1 - 21)	12 (0 - 50) 29 (0 - 67)	88 (50 - 100) 71 (33 - 100)
Local Search (steepest, greedy, two_nodes)	15208 (14950 - 16536) 15141 (14950 - 15941)	2 (0 - 6) 2 (0 - 8)	30 (0 - 60) 29 (0 - 50)	30 (0 - 50) 29(0 - 62)
Local Search (steepest, random, two_edges)	1196022 (974681 - 1465351) 1205847 (959433 - 1439920)	132 (118 - 148) 134 (119 - 160)	41 (35 - 51) 41 (34 - 49)	59 (49 - 65) 59 (51 - 66)
Local Search (steepest, random, two_nodes)	56669 (42121 - 75795) 54195 (42722 - 70253)	101 (70 - 147) 97 (69 - 167)	39 (26 - 53) 40 (26 - 52)	61 (47 - 74) 60(48 - 74)

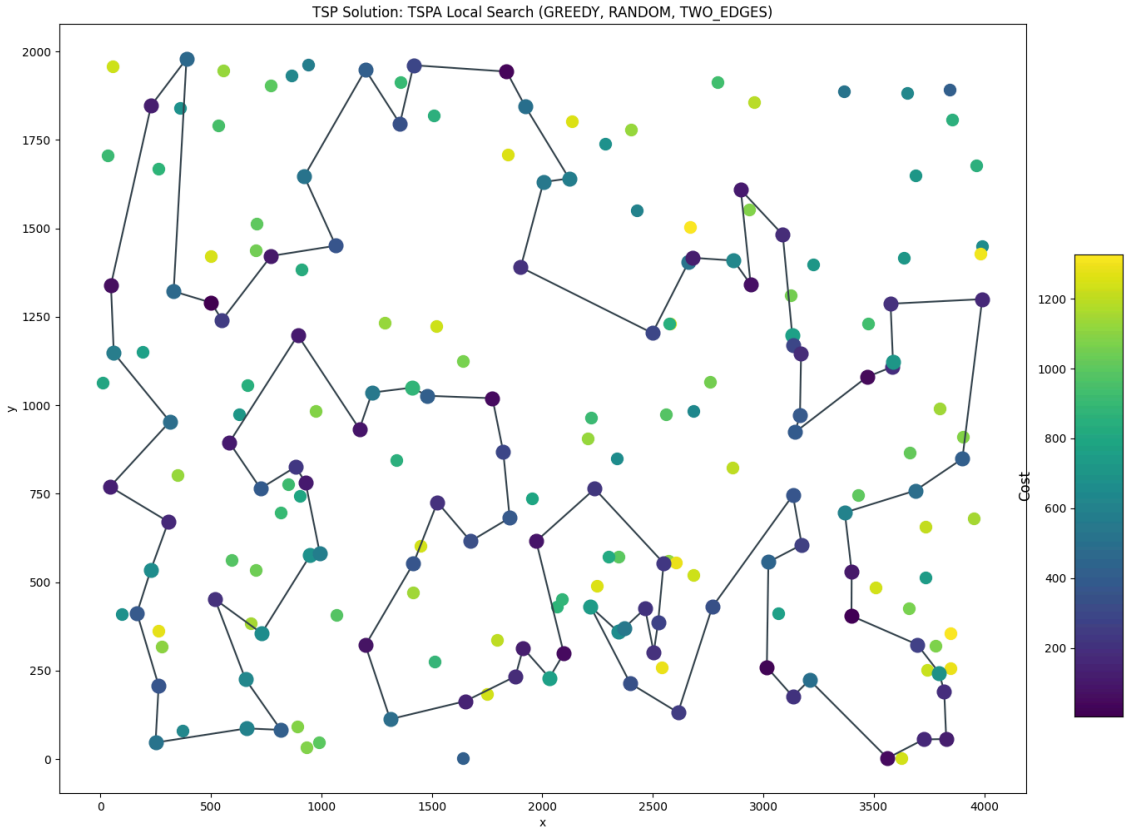
Best solutions

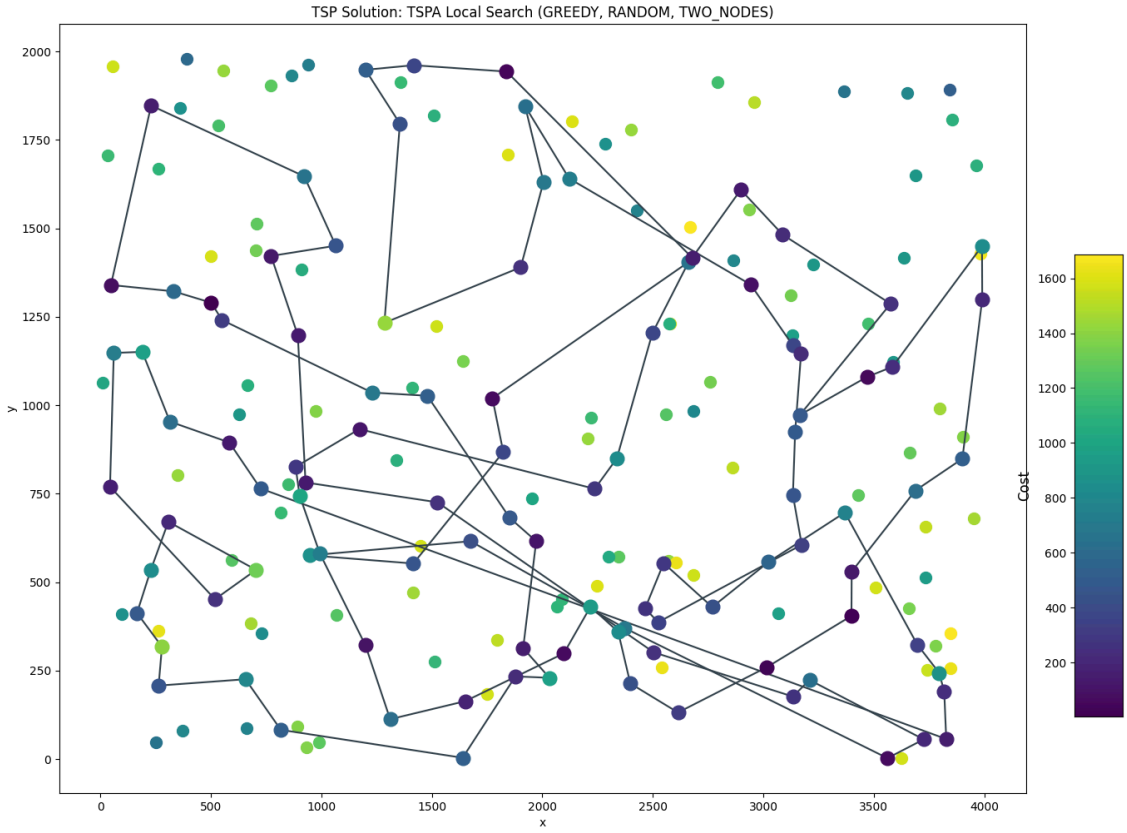
Note: additional cost is depicted using a color scale

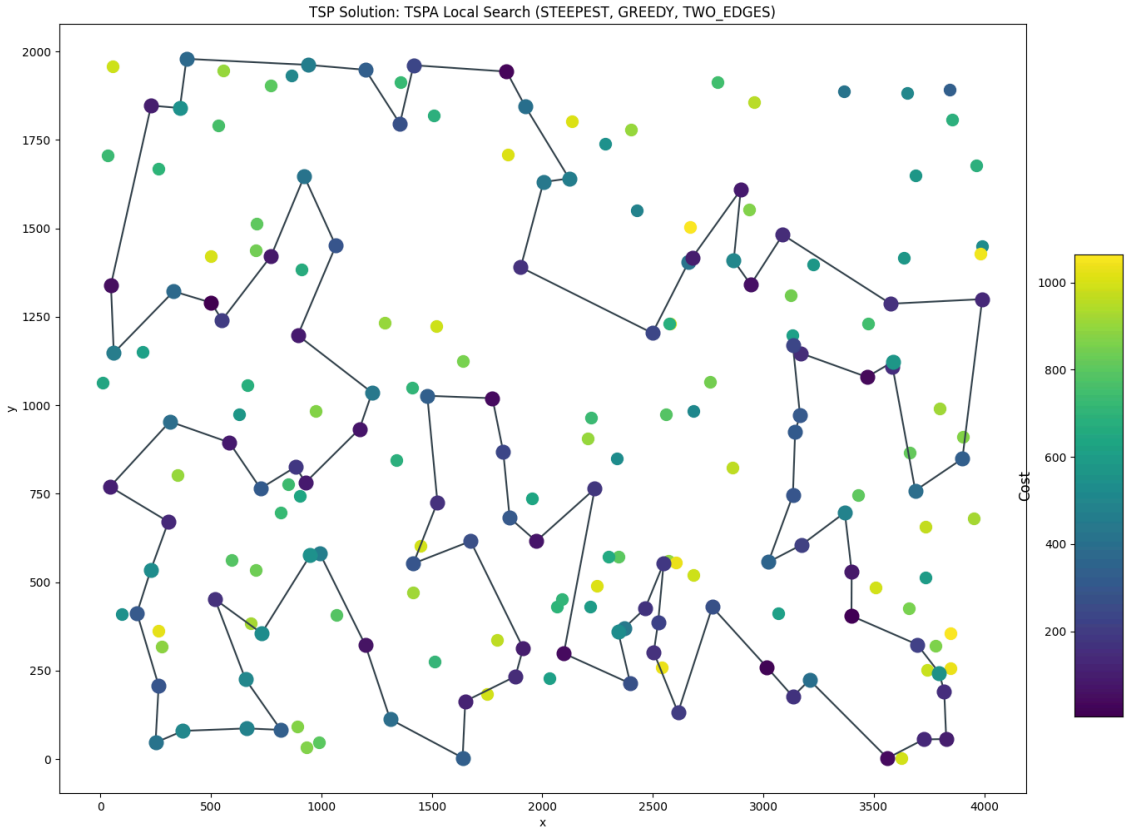
TSPA

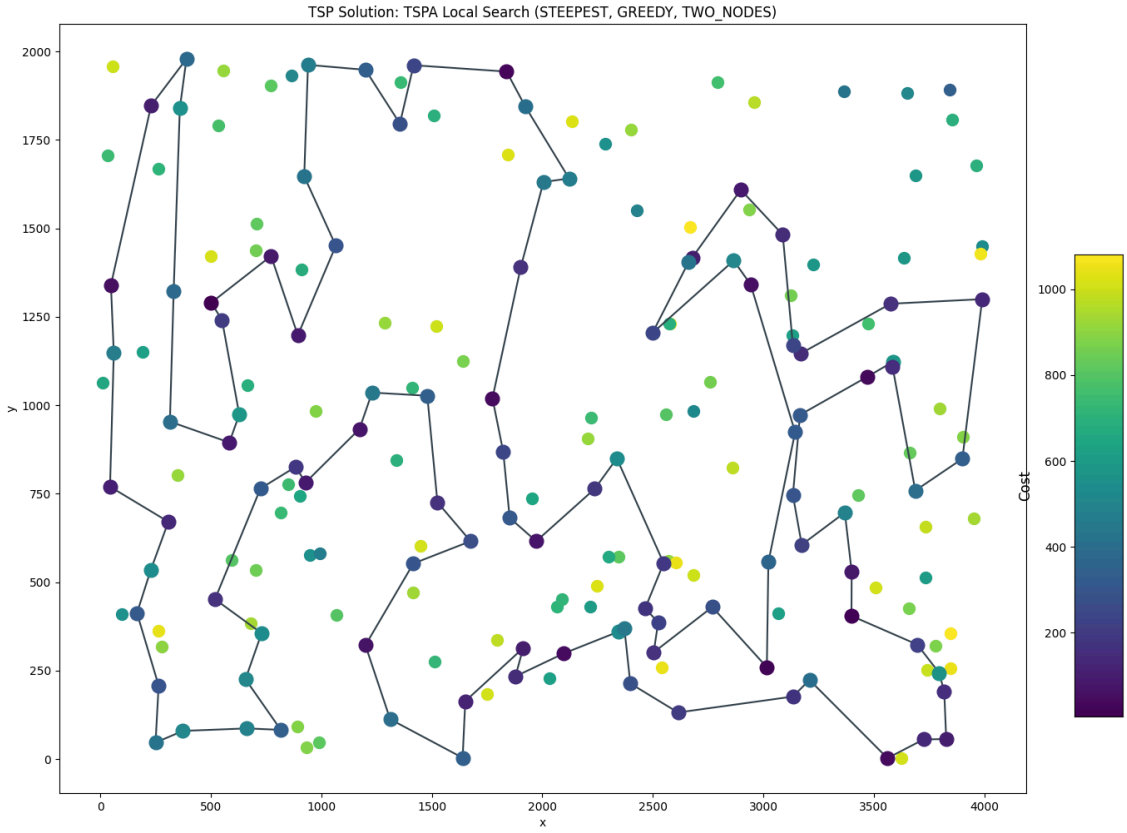


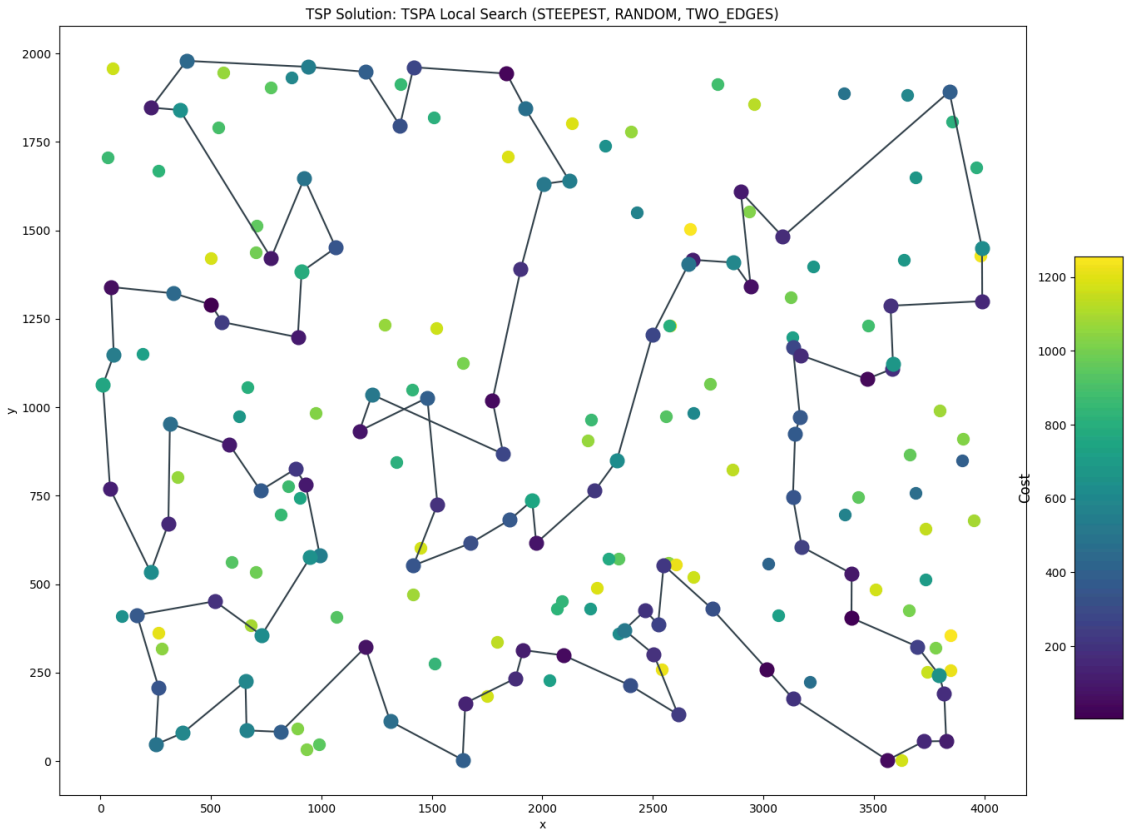

TSP Solution: TSPA Local Search (GREEDY, GREEDY, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
1000
800
600
400
200
Cost

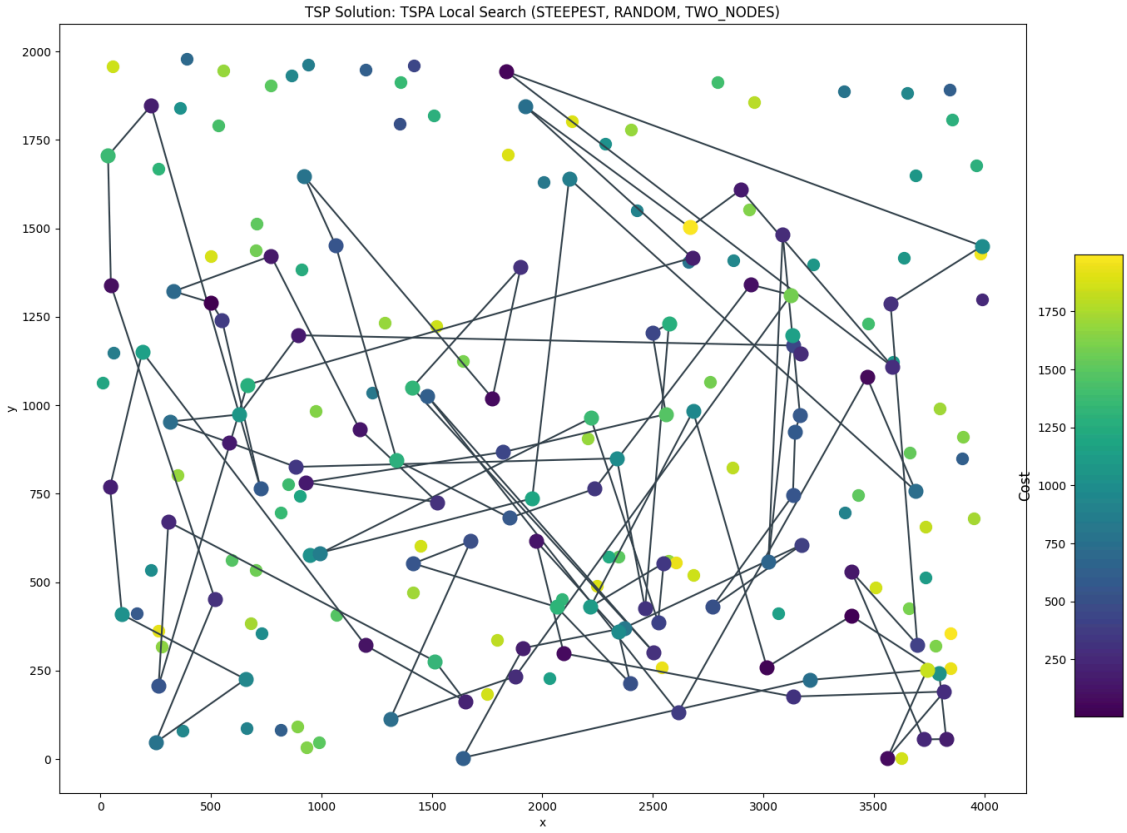

TSP Solution: TSPA Local Search (GREEDY, RANDOM, TWO_EDGES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
1200
1000
800
600
400
200
Cost


TSP Solution: TSPA Local Search (GREEDY, RANDOM, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
Cost
1600
1400
1200
1000
800
600
400
200

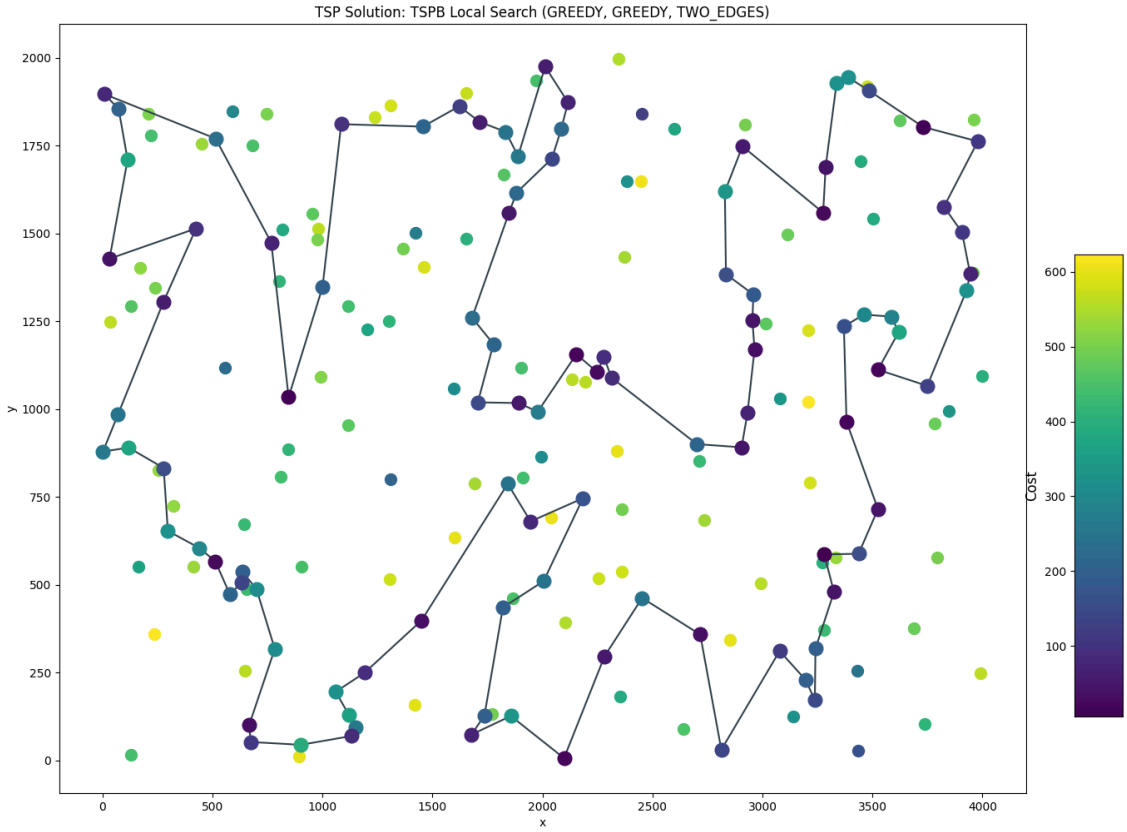

TSP Solution: TSPA Local Search (STEEPEST, GREEDY, TWO_EDGES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
1000
800
600
400
200
Cost

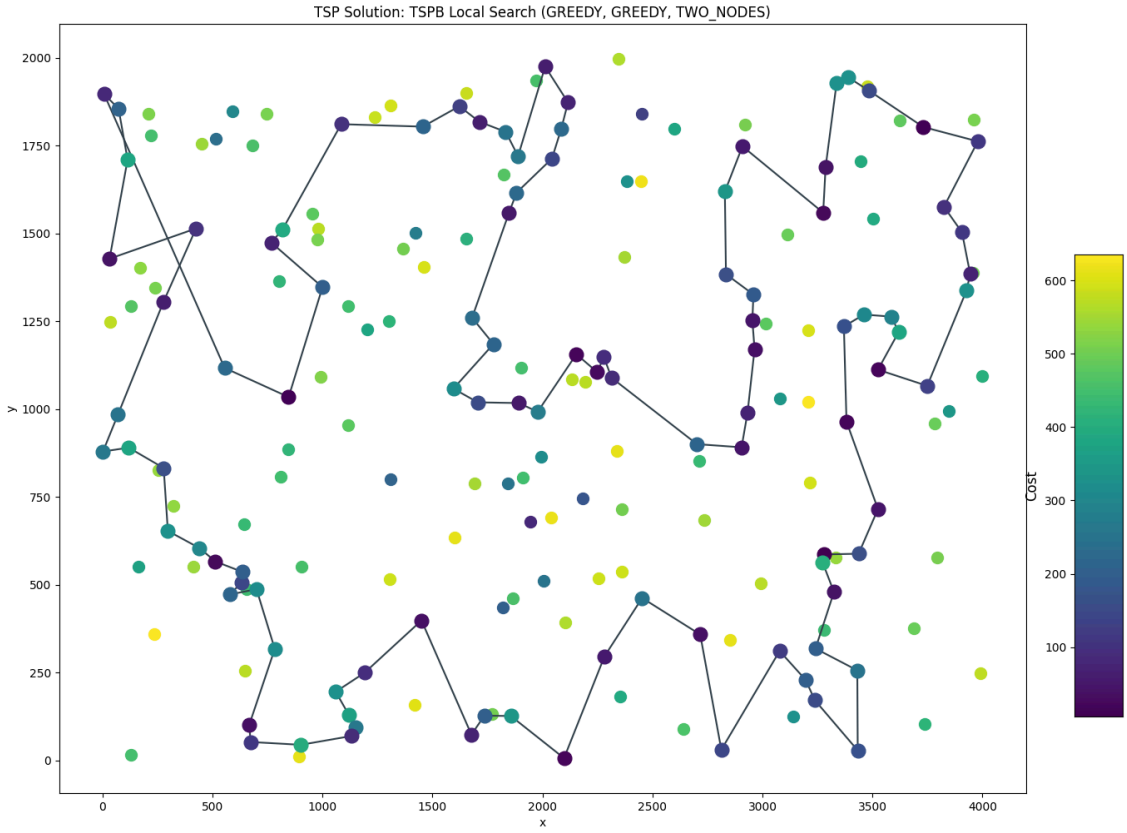

TSP Solution: TSPA Local Search (STEEPEST, GREEDY, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
1000
800
600
400
200
Cost

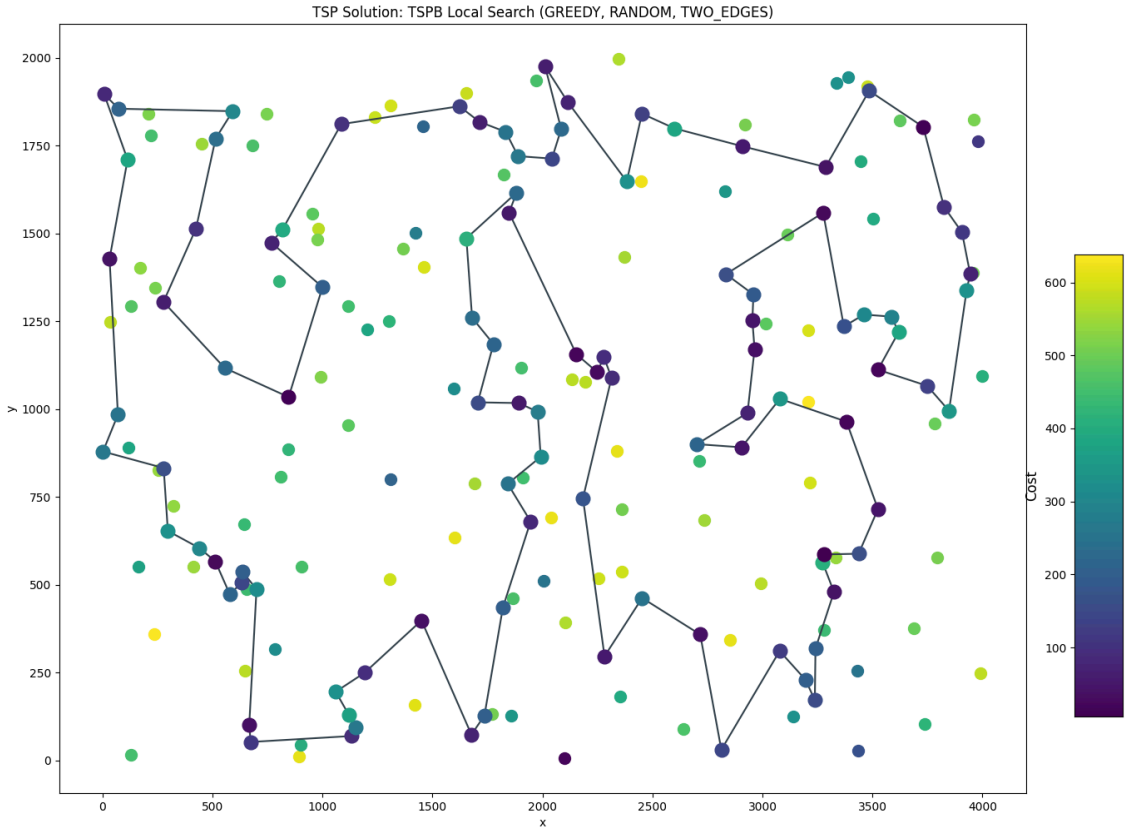

TSP Solution: TSPA Local Search (STEEPEST, RANDOM, TWO_EDGES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
1200
1000
800
600
400
200
Cost

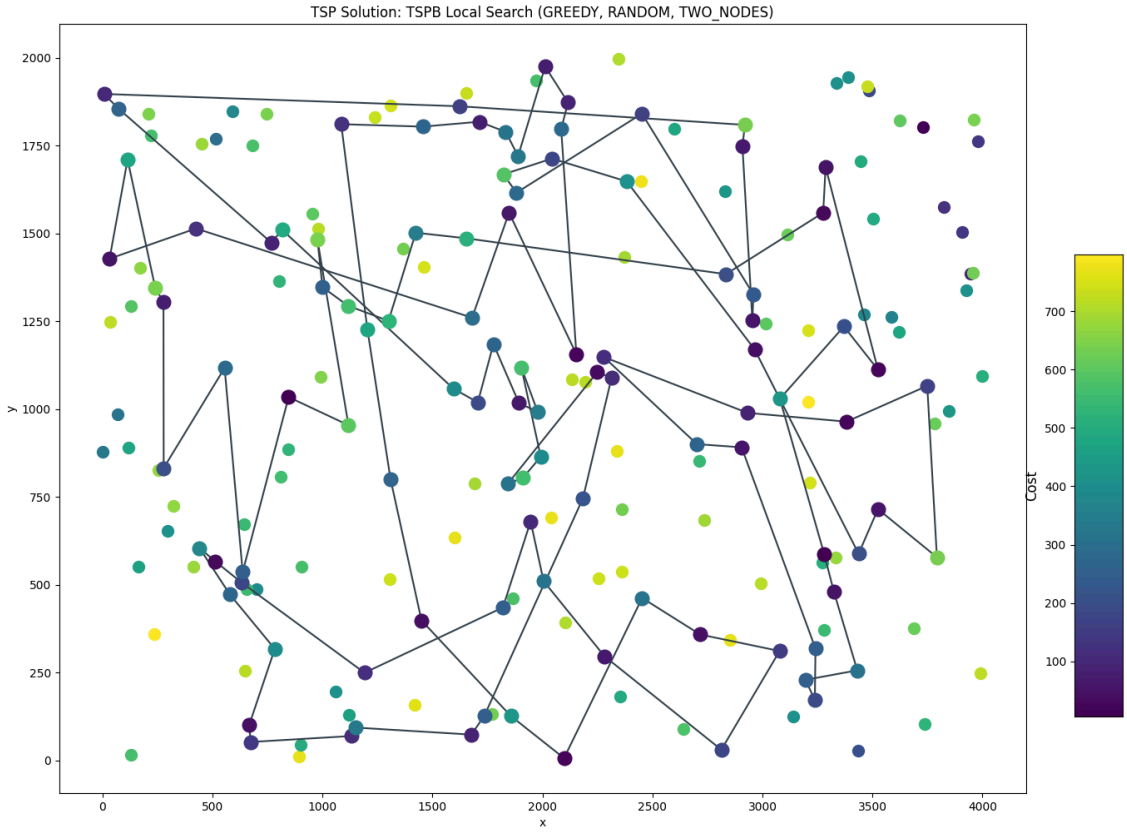

TSP Solution: TSPA Local Search (STEEPEST, RANDOM, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
Cost
1750
1500
1250
1000
750
500
250

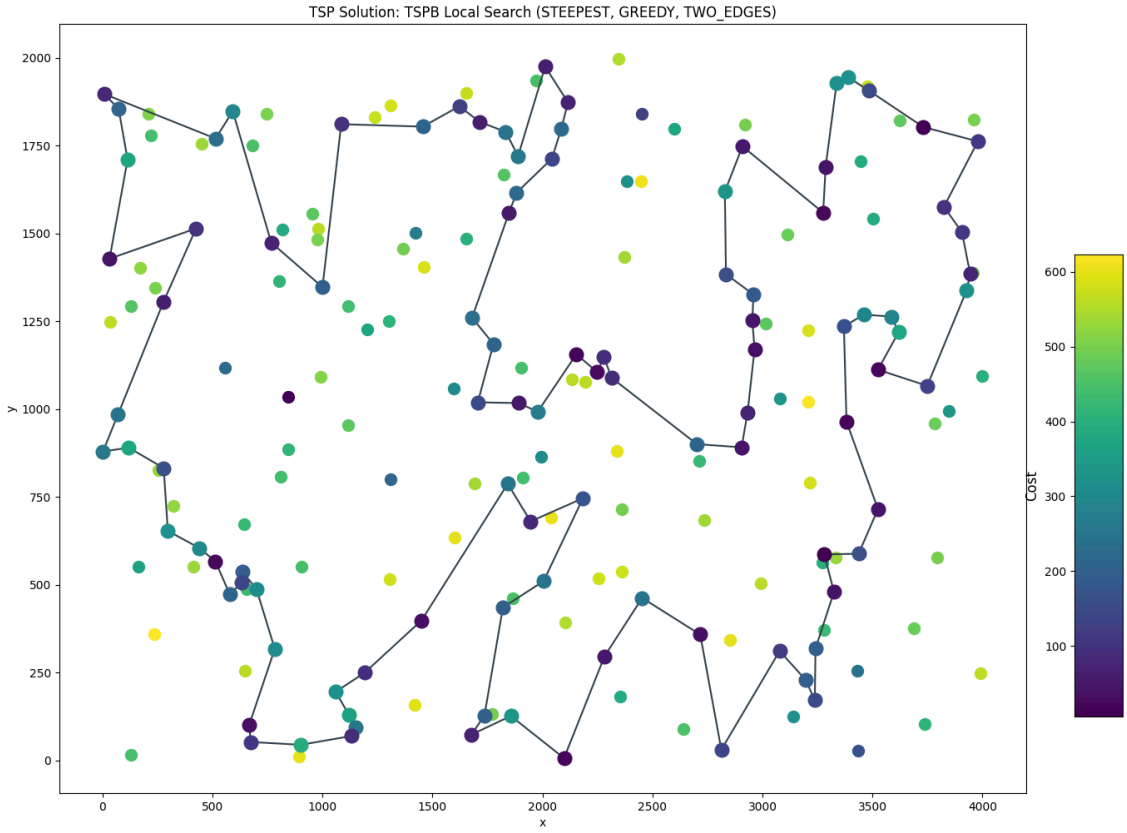
TSPB

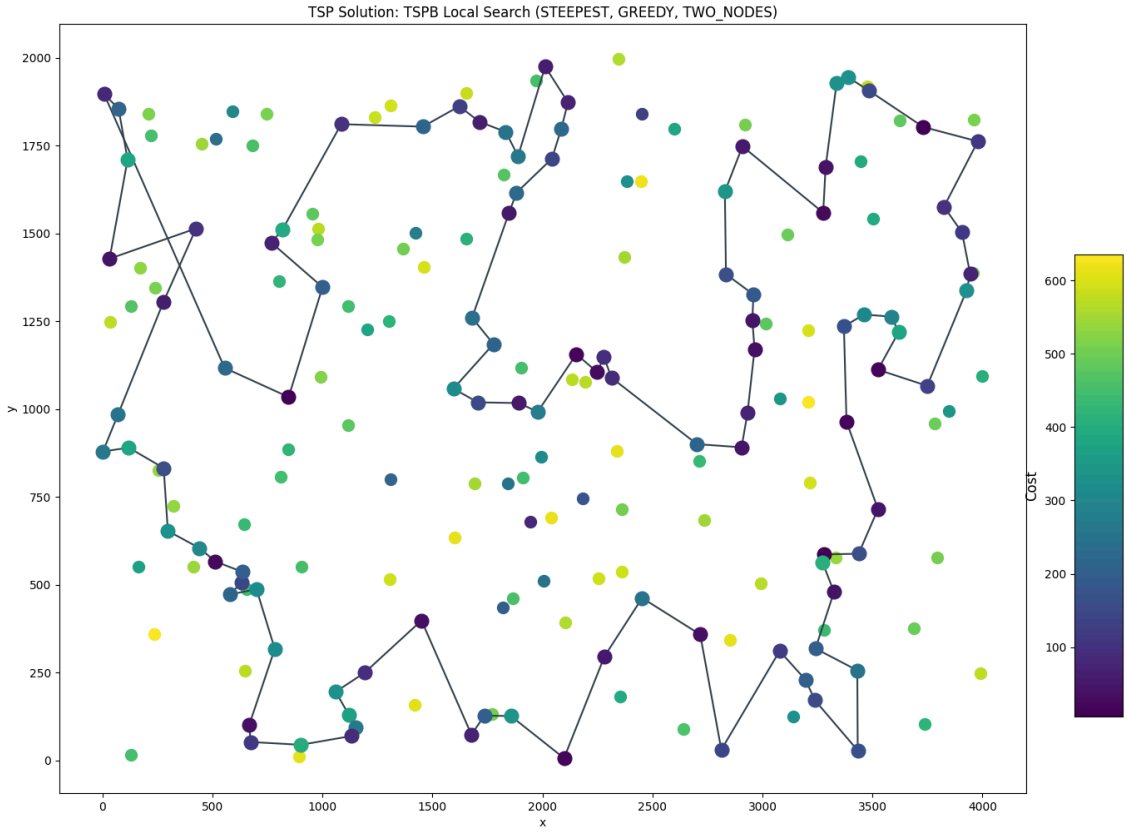

TSP Solution: TSPB Local Search (GREEDY, GREEDY, TWO_EDGES)
y
x
Cost
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500
4000
100
200
300
400
500
600

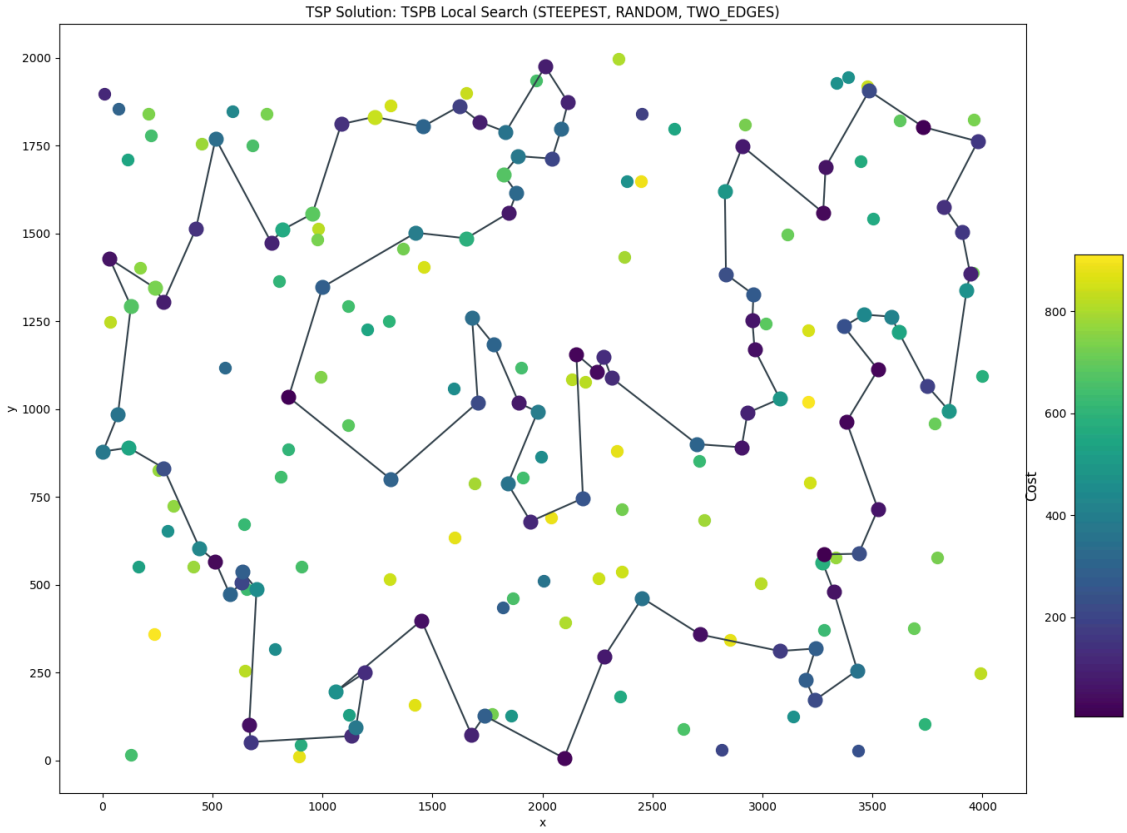

TSP Solution: TSPB Local Search (GREEDY, GREEDY, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
600
500
400
300
200
100
Cost

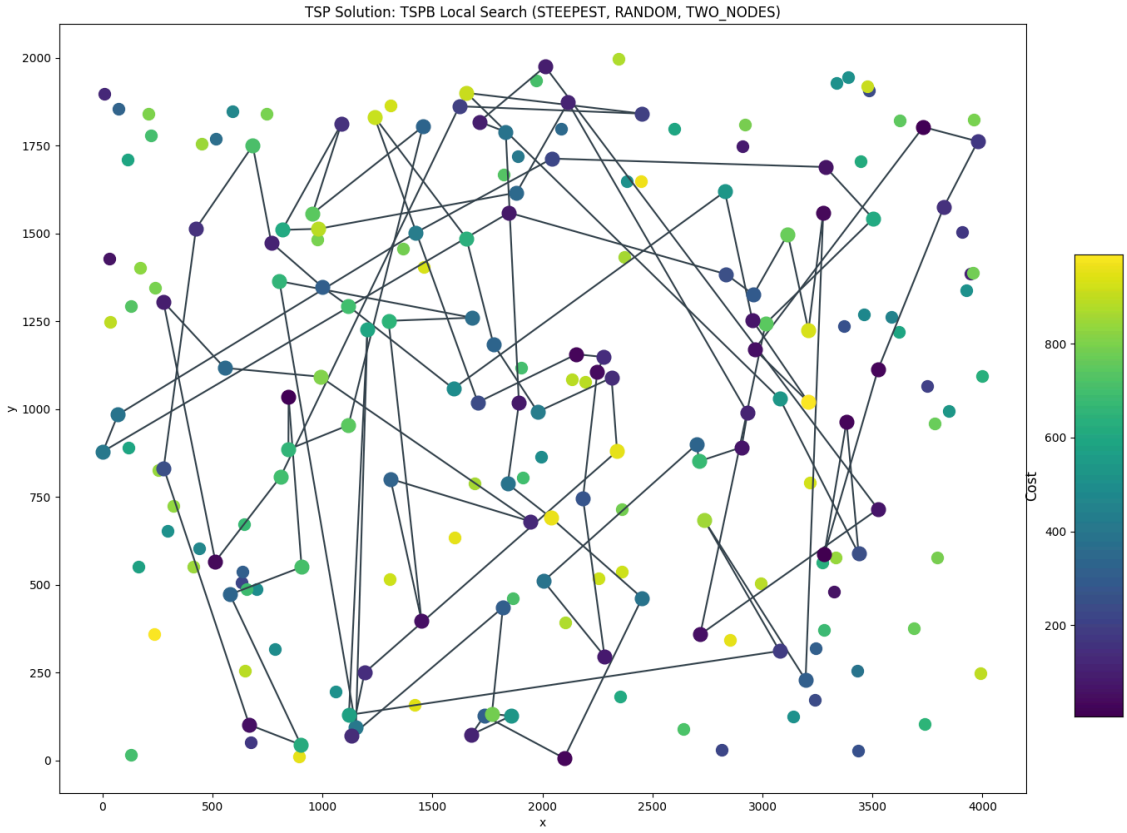

TSP Solution: TSPB Local Search (GREEDY, RANDOM, TWO_EDGES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
600
500
400
300
200
100
Cost


TSP Solution: TSPB Local Search (GREEDY, RANDOM, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
Cost
700
600
500
400
300
200
100


TSP Solution: TSPB Local Search (STEEPEST, GREEDY, TWO_EDGES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
600
500
400
300
200
100
Cost


TSP Solution: TSPB Local Search (STEEPEST, GREEDY, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
600
500
400
300
200
100
Cost


TSP Solution: TSPB Local Search (STEEPEST, RANDOM, TWO_EDGES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
800
600
400
200
Cost


TSP Solution: TSPB Local Search (STEEPEST, RANDOM, TWO_NODES)
2000
1750
1500
1250
1000
750
500
250
0
y
0
500
1000
1500
2000
2500
3000
3500
4000
x
800
600
400
200
Cost

Conclusions

- Local Search (LS) is a post-processing method that is guaranteed to yield solution at least as good as the initial one
- All LS configurations significantly improved the performance of initially random (and horrendous) solution
    - besides intra move two nodes exchange, the performance was comparable to that of the best heuristics considered so far
- Intra move two edges exchange neighborhood is significantly higher than that of intra move two nodes exchange
    - Hence, poor performance of random starting solution with two nodes exchange - these moves rely on decently good edges, and tend to create crossing paths (proven to be non-optimal)
- Performance of Greedy and Steepest LS type is comparable for all configurations
    - Greedy seems more explorative - and thus less likely to get stuck at local optima due to randomness
    - Steepest seems more exploitative
    - Greedy seems to perform slightly better in TSPB - this may be because exploration is preferred when node cost differences are not that high
    - Steepest seems to perform slightly better in TSPA - this may be because exploitation is preferred when node cost differences are significantly high
    - Besides random initial solution + intra move two nodes exchange - for this configuration, greedy performs better overall
- Starting from random solution generally takes (often significantly!) longer than starting from greedy - which is expected, since there's significantly more improving moves to be done than in already good solution
    - In case of greedy start, LS is more of a "fine refinement", trying to squeeze out at least just a little more by making a few moves - which seems appropriate use of LS
    - In case of random, LS is responsible for actually making the solution good - hence number of total moves in hundreds