Authors: Weronika Zawadzka 151943 Eliza Czaplicka 151963

Date: December 08, 2024

Best solutions have been checked with the solution checker

Summary Report for w7_large_neigh_search

Based on methods from [our github project]

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.

Pseudocode of implemented algorithms

Stopping condition = average running time of MSLS from the previous assignment

Repair operator = Greedy2RegretWeightedSum

Local search = LS_Steepest2edgesMoveEvals

Main function

Generate an initial solution x (random solution)

x := Local search (x)

while (System.currentTimeMillis() - startingTime < maxTime) {

    y := Destroy (x)

    y := Repair (y) [Greedy2RegretWeightedSum(y)]

    y := Local search (y) (optional)

    currF = evaluate(y)

If currF < bestF then x := y

}

Destroy() operator

```
n_subpaths = random choice: 2 or 3 subpaths

percentage = randomly choose between 20-30% nodes

length_subpath = (solution.size() * percentage) / 100 / n_subpaths
```

Step 1: Evaluate all subpaths of length length_subpath in solution (obj. F)

Step 2: Create probabilities: normalize scores based on the highest score

Step 3: Randomly select n_subpaths (2-3) subpaths based on probabilities

Step 4: Remove those subpaths from solution

Step 5: Remove random nodes if not enough nodes were removed

(if subpaths were overlapping)

Step 7: Return the destroyed solution

Summary performance of each method

Summary for Execution Time (ms)

Method	A	B
Greedy2RegretWeightedSumMethod	3.745 (2 - 42)	3.665 (0 - 36)
LS_Greedy2edgesGreedy	29.83 (7 - 113)	38.725 (12 - 127)
LS_Greedy2edgesRandom	106.065 (58 - 297)	105.985 (56 - 220)
LS_Greedy2nodesGreedy	55.41 (13 - 249)	60.825 (17 - 237)
LS_Greedy2nodesRandom	296.865 (108 - 616)	126.165 (60 - 334)
IteratedLocalSearch	13174.7 (13172 - 13181)	13174.65 (13172 - 13186)
LargeNeighborhoodSearch	13470.85 (13439 - 13746)	13478.7 (13439 - 13800)
LargeNeighborhoodSearchLS	13446.5 (13431 - 13459)	13449.05 (13437 - 13462)
MultipleStartLocalSearch	13383.2 (12157-14328)	13172.15 (12284-14056)
LS_Steepest2edgesMoveEvals	87.105 (41 - 291)	83.625 (41 - 321)
LS_Steepest2edgesRandom	924.43 (621 - 1148)	926.84 (788 - 1084)
LS_Steepest2nodesGreedy	96.655 (57 - 291)	131.26 (84 - 394)
LS_Steepest2nodesRandom	1592.99 (1126 - 3193)	1642.05 (1184 - 2385)

Summary for Objective Function Value

Method	A	B
Greedy2RegretWeightedSumMethod	72123.08 (71108 - 73321)	50904.245 (47144 - 55700)
LS_Greedy2edgesGreedy	71798.75 (71612 - 71910)	51770.925 (50579 - 51955)
LS_Greedy2edgesRandom	75071.865 (72146 - 78848)	49265.74 (46899 - 51687)
LS_Greedy2nodesGreedy	71643.38 (71624 - 71675)	51822.39 (51728 - 51931)
LS_Greedy2nodesRandom	85784.375 (79581 - 94992)	60971.66 (53818 - 70256)
IteratedLocalSearch	69615.25 (69239 - 69951)	**43879.4** (43456 - 44527)
LargeNeighborhoodSearch	**69438.5** (69327 - 70034)	44103.75 (43897 - 44223)
LargeNeighborhoodSearchLS	**69321.4** (69207 - 69501)	**43965.4** (43829 - 44106)
MultipleStartLocalSearch	72616.2 (72053 - 73027)	46813.0 (46123 - 47377)
LS_Steepest2edgesGreedy	71865.0 (71865 - 71865)	51790.0 (51790 - 51790)
Steepest2edgesMoveEvals	75895.485 (72746 - 80024)	50031.255 (46114 - 53809)
LS_Steepest2edgesRandom	75337.8 (72622 - 78771)	49424.465 (46854 - 52233)
LS_Steepest2nodesGreedy	71624.0 (71624 - 71624)	51865.0 (51865 - 51865)
LS_Steepest2nodesRandom	88075.565 (81006 - 99501)	63187.48 (54443 - 71877)

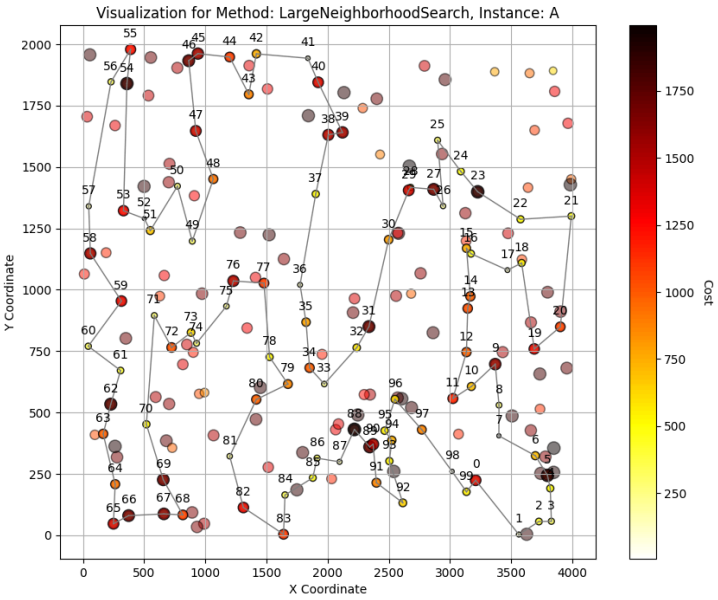
Summary for Iteration Count

Method	A	B
IteratedLocalSearch	3040.8 (2576.0 - 3377.0)	3468.8 (3089.0 - 3743.0)
LargeNeighborhoodSearch	4621.3 (4492 - 4742)	4638.95 (4488 - 4738)
LargeNeighborhoodSearchLS	2831.3 (2685 - 2959)	2866.05 (2639 - 2977)

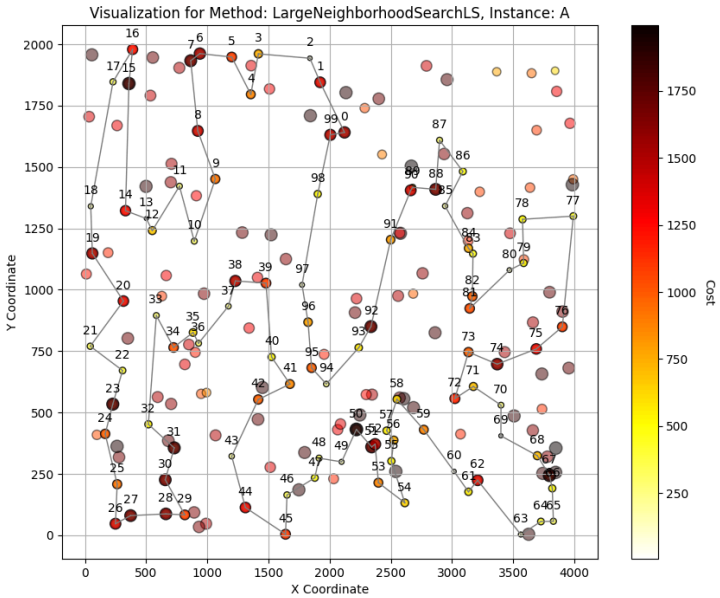
2D visualisations of best solutions

Instance: A

Method: LargeNeighborhoodSearch with score=69327

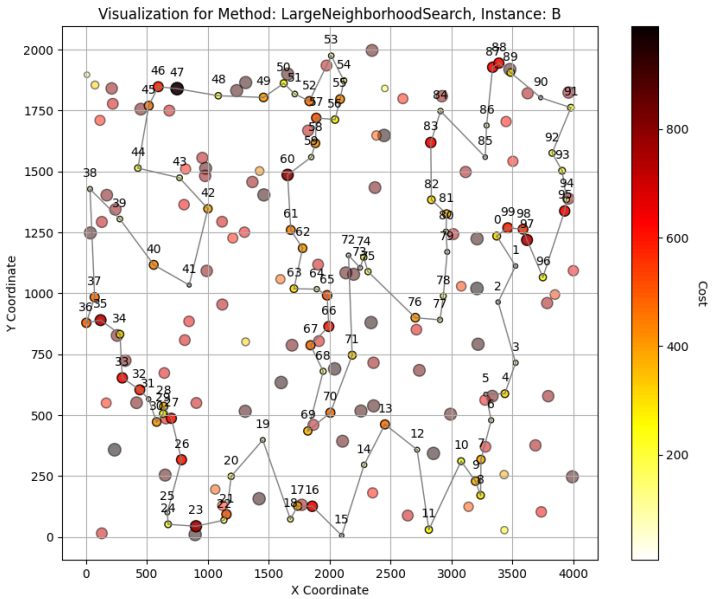


Method: LargeNeighborhoodSearchLS with score=69207

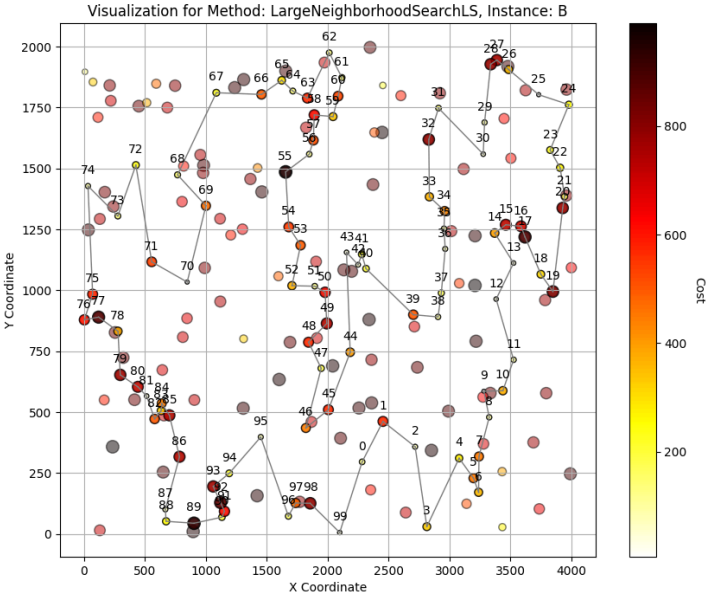


Instance: B

Method: LargeNeighborhoodSearch with score=44001



Method: LargeNeighborhoodSearchLS with score=43829



Best solutions, indices

Instance: A

Method: LargeNeighborhoodSearch

Lowest Objective Function Value (f_val): 69327

Solution:

25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 92, 129, 57, 55, 52, 178, 106, 185, 40, 196, 81, 90, 165, 138, 14, 144, 49, 102, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 186, 89, 183, 143, 0, 117, 93, 140, 68, 46, 115, 139, 41, 193, 159, 69, 108, 18, 22, 146, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 184, 42, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44

Method: LargeNeighborhoodSearchLS

Lowest Objective Function Value (f_val): 69207

Solution:

186, 89, 183, 143, 0, 117, 93, 140, 68, 46, 115, 139, 41, 193, 159, 69, 108, 18, 22, 146, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 35, 184, 42, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 92, 129, 57, 179, 196, 81, 90, 165, 40, 185, 55, 52, 106, 178, 49, 14, 144, 102, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23

Instance: B

Method: LargeNeighborhoodSearch

Lowest Objective Function Value (f_val): 44001

Solution:

95, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 177, 5, 78, 175, 162, 80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 38, 63, 135, 131, 121, 51, 90, 122, 133, 10, 115, 147, 6, 188, 169, 132, 70, 3, 15, 145, 13, 195, 168, 43, 139, 11, 138, 33, 160, 144, 104, 8, 21, 82, 111, 29, 0, 109, 35, 143, 106, 124, 62, 18, 55, 34, 170, 152, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 179, 22, 99, 130

Method: LargeNeighborhoodSearchLS

Lowest Objective Function Value (f_val): 43829

Solution:

77, 81, 153, 187, 163, 89, 127, 103, 113, 176, 194, 166, 86, 185, 95, 130, 99, 22, 179, 172, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 143, 35, 109, 0, 29, 111, 82, 21, 8, 104, 144, 160, 33, 138, 11, 139, 43, 168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6, 147, 90, 51, 121, 131, 122, 135, 63, 38, 27, 16, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 162, 175, 78, 142, 45, 5, 177, 36, 61, 91, 141

Conclusions

Large Neighborhood Search with Local Search appears to be the most effective method for minimizing the objective function yet (on instance A) and very close to being best on instance B as well. **Large Neighborhood Search**, so without a local search after repair operator, performs similarly but is slightly less effective. On instance B however, ILS was still the best. **ILS** could search more quickly than LNS due to its simpler approach and smaller neighborhood, allowing it to potentially reach better solutions faster and this speed advantage could explain why ILS had a slightly better score in this case. When it comes to the number of iterations, ILS without LS has managed to do 50% more than the version with LS.