Authors: Weronika Zawadzka 151943 Eliza Czaplicka 151963

Date: October 26, 2024

Best solutions have been checked with the solution checker

Summary Report for w3_local_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.

What kind of randomization was used - initial solution

We initialized the Random object in Java with a fixed seed value (using RANDOM = new Random(222)). Then the initial random solution was created by first shuffling list of all possible ids of nodes, and taking first 50%.

Pseudocode of implemented algorithms

- **Greedy 2 nodes Greedy**

```
Generate a starting solution by greedy 2 regret weighted sum algorithm.
initialCost = total distance + cost of nodes
while True:
    for node1 in current solution in random order:
        for node2 in all nodes in random order:
            if node2 in current solution:
                swap nodes
            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<initialCost:
                current solution = new solution
                initialCost = cost
                break
    if the solution has not changed:
        flag = False
return current solution
```

- **Greedy 2 nodes Random**

```
Generate a starting solution randomly shuffling the nodes and selecting the first half.
initialCost = total distance + cost of nodes
while True:
    for node1 in current solution in random order:
        for node2 in all nodes in random order:
            if node2 in current solution:
                swap nodes
            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<initialCost:
                current solution = new solution
                initialCost = cost
                break
    if the solution has not changed:
        flag = False
return current solution
```

- **Steepest 2 nodes Greedy**

```
Generate a starting solution by greedy 2 regret weighted sum algorithm.
initialCost = total distance + cost of nodes
while True:
    bestSolutionFound
    bestCost
    for node1 in current solution:
        for node2 in all nodes:
            if node2 in current solution:
                swap nodes
            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<bestCost:
                bestSolutionFound = new solution
                bestCost = cost
    if bestCost<initialCost:
        current solution = bestSolutionFound
        initialCost = bestCost
    else:
        flag = False
return current solution
```

- **Steepest 2 nodes Random**

```
Generate a starting solution randomly shuffling the nodes and selecting the first half.
initialCost = total distance + cost of nodes
while True:
    bestSolutionFound
    bestCost
    for node1 in current solution:
        for node2 in all nodes:
            if node2 in current solution:
                swap nodes
            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<bestCost:
                bestSolutionFound = new solution
                bestCost = cost
    if bestCost<initialCost:
        current solution = bestSolutionFound
        initialCost = bestCost
    else:
        flag = False
return current solution
```

- **Greedy 2 edges Greedy**

```
Generate a starting solution by greedy 2 regret weighted sum algorithm.
initialCost = total distance + cost of nodes
while True:
    for node1 in current solution in random order:
        for node2 in all nodes in random order:
            if node2 in current solution:
                if next to each other then skip
                reverse order between smaller & larger id
            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<initialCost:
                current solution = new solution
                initialCost = cost
                break
    if the solution has not changed:
        flag = False
return current solution
```

- **Greedy 2 edges Random**

```
Generate a starting solution randomly shuffling the nodes and selecting the first half.
initialCost = total distance + cost of nodes
while True:
    for node1 in current solution in random order:
        for node2 in all nodes in random order:
            if node2 in current solution:
                if next to each other then skip
                reverse order between smaller & larger id
            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<initialCost:
                current solution = new solution
                initialCost = cost
                break
    if the solution has not changed:
        flag = False
return current solution
```

- **Steepest 2 edges Greedy**

```
Generate a starting solution by greedy 2 regret weighted sum algorithm.
initialCost = total distance + cost of nodes
while True:
    bestSolutionFound
    bestCost
    for node1 in current solution:
        for node2 in all nodes:
            if node2 in current solution:
                if next to each other then skip
                reverse order between smaller & larger id

            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<bestCost:
                bestSolutionFound = new solution
                bestCost = cost
    if bestCost<initialCost:
        current solution = bestSolutionFound
        initialCost = bestCost
    else:
        flag = False
return current solution
```

- **Steepest 2 edges Random**

```
Generate a starting solution randomly shuffling the nodes and selecting the first half.
initialCost = total distance + cost of nodes
while True:
    bestSolutionFound
    bestCost
    for node1 in current solution:
        for node2 in all nodes:
            if node2 in current solution:
                if next to each other then skip
                reverse order between smaller & larger id
            else:
                change node 1 for node 2
            cost = new total distance + new cost of nodes
            if  cost<bestCost:
                bestSolutionFound = new solution
                bestCost = cost
    if bestCost<initialCost:
        current solution = bestSolutionFound
        initialCost = bestCost
    else:
        flag = False
return current solution
```

Summary performance of each method

Summary for Execution Time (ms)

Method	**A**	**B**
Greedy2RegretMethod	3.89 (0 - 36)	4.035 (0 - 47)
Greedy2RegretWeightedSumMethod	3.745 (2 - 42)	3.665 (0 - 36)
Greedy2edgesGreedy	29.83 (7 - 113)	38.725 (12 - 127)
Greedy2edgesRandom	106.065 (58 - 297)	105.985 (56 - 220)
Greedy2nodesGreedy	55.41 (13 - 249)	60.825 (17 - 237)
Greedy2nodesRandom	296.865 (108 - 616)	126.165 (60 - 334)
GreedyCycleMethod	9.21 (2 - 91)	9.315 (1 - 109)
NearNeighborEndMethod	0.145 (0 - 16)	0.125 (0 - 6)
NearNeighborMethod	0.13 (0 - 8)	0.18 (0 - 14)
RandomMethod	0.12 (0 - 5)	0.08 (0 - 1)
Steepest2edgesGreedy	42.38 (32 - 134)	88.435 (73 - 182)
Steepest2edgesRandom	1230.58 (1070 - 1414)	1141.46 (1017 - 1339)
Steepest2nodesGreedy	96.655 (57 - 291)	131.26 (84 - 394)
Steepest2nodesRandom	1592.99 (1126 - 3193)	1654.205 (1184 - 2607)

The most consuming methods were the ones with random starting order - we see if we used greedy regret weighted sum heuristic as starting solution, the time decreased significantly. So that thoughtful starting points can reduce computation significantly. Overall, the methods from this week (starting with Steepest2… & Greedy2…) were on average more time consuming than other weeks' methods, but greedy methods were faster than steepest (since they accept first improving solution).

Summary for Objective Function Value

Method	A	B
Greedy2RegretMethod	116066.94 (105692 - 126951)	72783.945 (68395 - 78406)
Greedy2RegretWeightedSumMethod	72123.08 (71108 - 73321)	50904.245 (47144 - 55700)
Greedy2edgesGreedy	71798.75 (71612 - 71910)	51770.925 (50579 - 51955)
Greedy2edgesRandom	75071.865 (72146 - 78848)	49265.74 (46899 - 51687)
Greedy2nodesGreedy	71643.38 (71624 - 71675)	51822.39 (51728 - 51931)
Greedy2nodesRandom	85784.375 (79581 - 94992)	60971.66 (53818 - 70256)
GreedyCycleMethod	72987.8 (71719 - 75803)	51568.455 (49001 - 57271)
NearNeighborEndMethod	85208.68 (83590 - 89433)	54359.255 (52319 - 59030)
NearNeighborMethod	187048.41 (171521 - 207728)	141517.085 (117926 - 161912)
RandomMethod	264976.085 (239099 - 289258)	213532.255 (189044 - 237132)
Steepest2edgesGreedy	71865.0 (71865 - 71865)	51790.0 (51790 - 51790)
Steepest2edgesRandom	75305.065 (73122 - 79322)	49511.875 (46569 - 53465)
Steepest2nodesGreedy	71624.0 (71624 - 71624)	51865.0 (51865 - 51865)
Steepest2nodesRandom	88075.565 (81006 - 99501)	63187.48 (54443 - 71877)

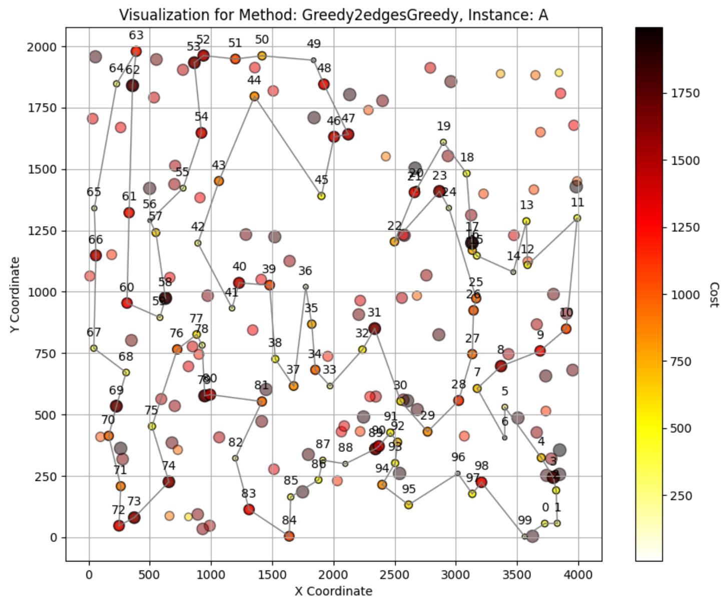
This week's methods turned out to be winners among all previous weeks in terms of average. Namely, steepest with greedy heuristic initialization (2regret weighted sum) and with intra methods of swapping 2 nodes won on instance A with lowest average score of 71624. However, lowest min - so the lowest score obtained overall was still achieved by greedy 2 regret weighted sum.

When it comes to instance B, the winner among average scores is greedy 2 edges method with random initialization with score of 49265.74. The method with the lowest min however was again achieved by steepest algorithm - here steepest 2 edges with random initialization. So greedy local search implementations were faster than steepest, but steepest achieved on average better scores.

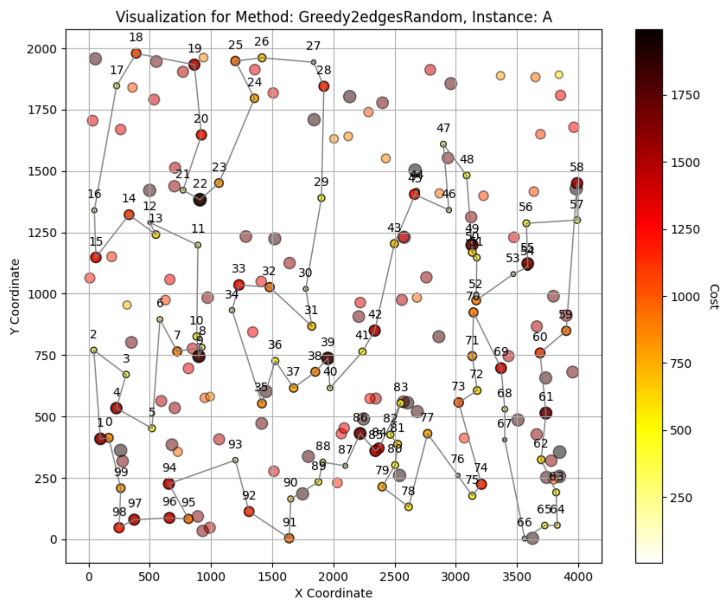
2D visualisations of best solutions

Instance: A

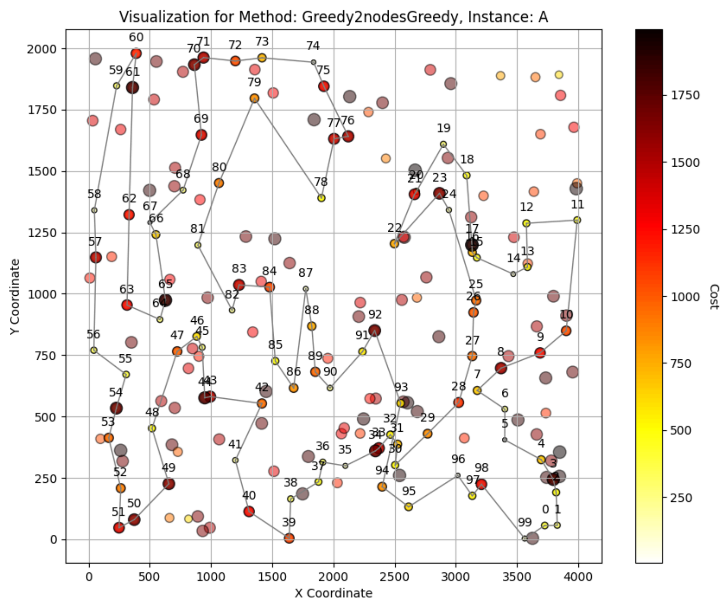
Method: Greedy2edgesGreedy with score=71612



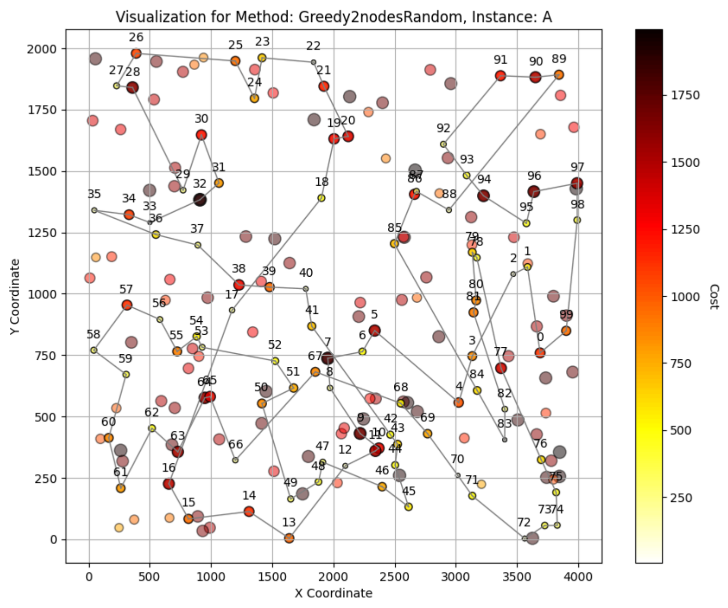
Method: Greedy2edgesRandom with score=72146



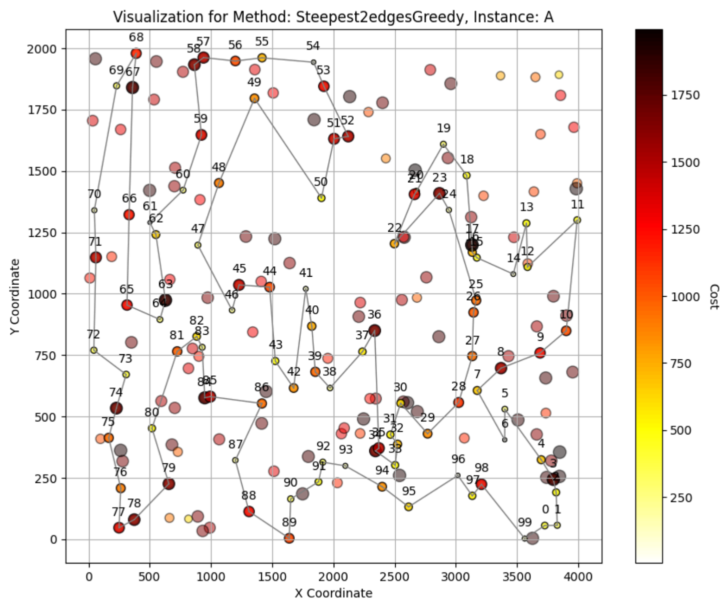
Method: Greedy2nodesGreedy with score=71624



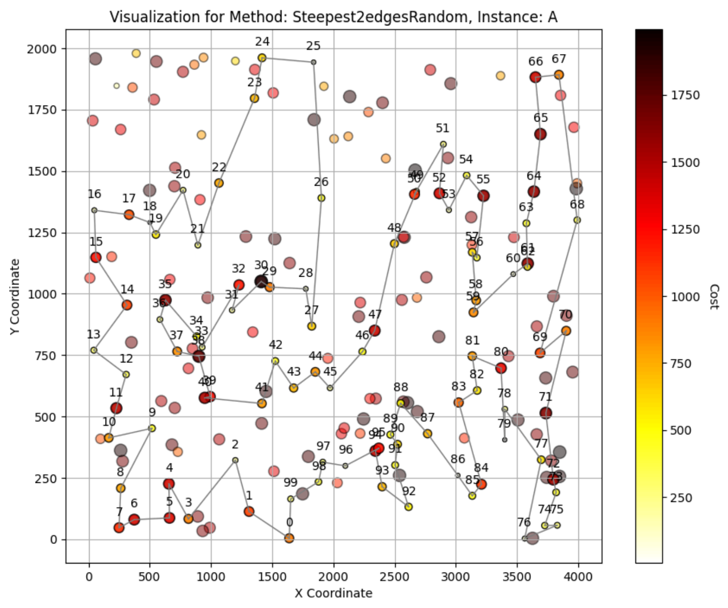
Method: Greedy2nodesRandom with score=79581



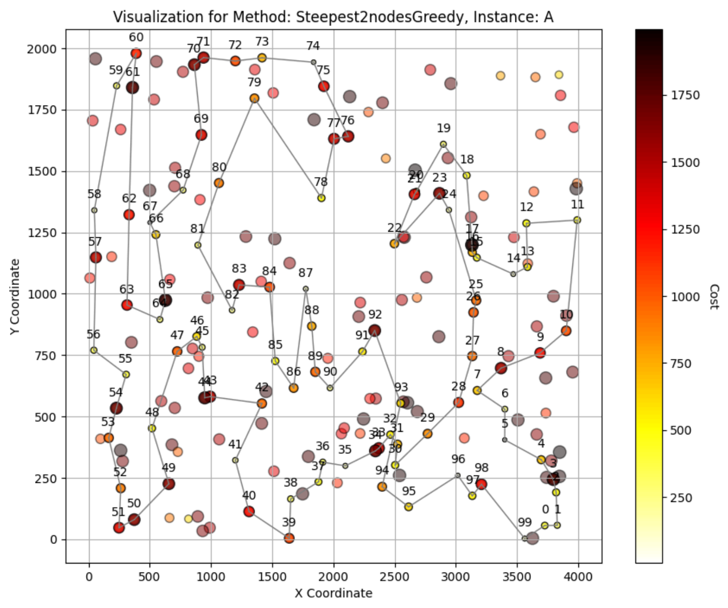
Method: Steepest2edgesGreedy with score=71865



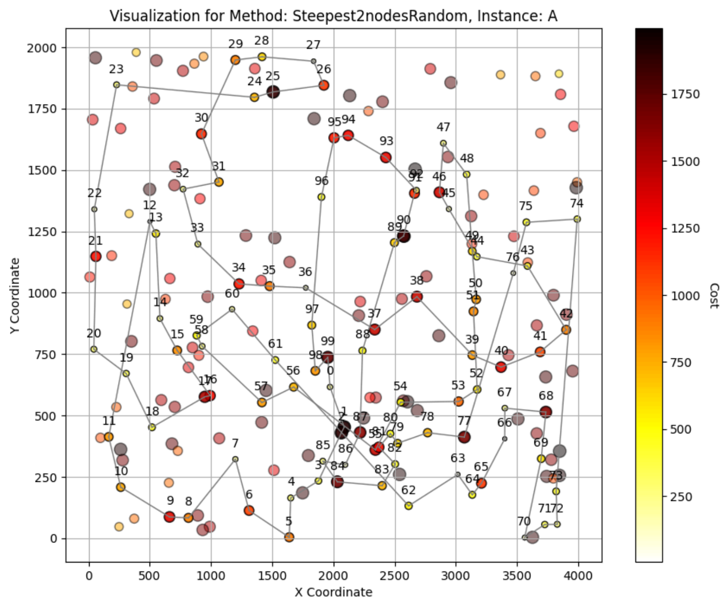
Method: Steepest2edgesRandom with score=73122



Method: Steepest2nodesGreedy with score=71624

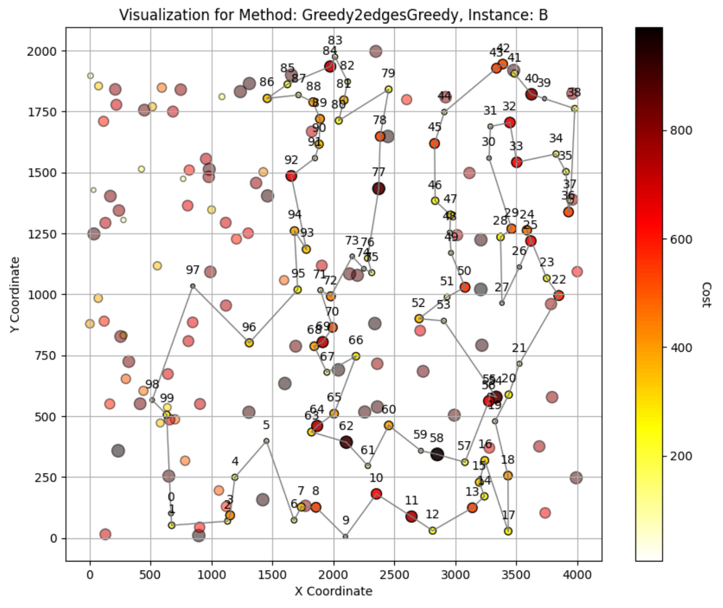


Method: Steepest2nodesRandom with score=81006

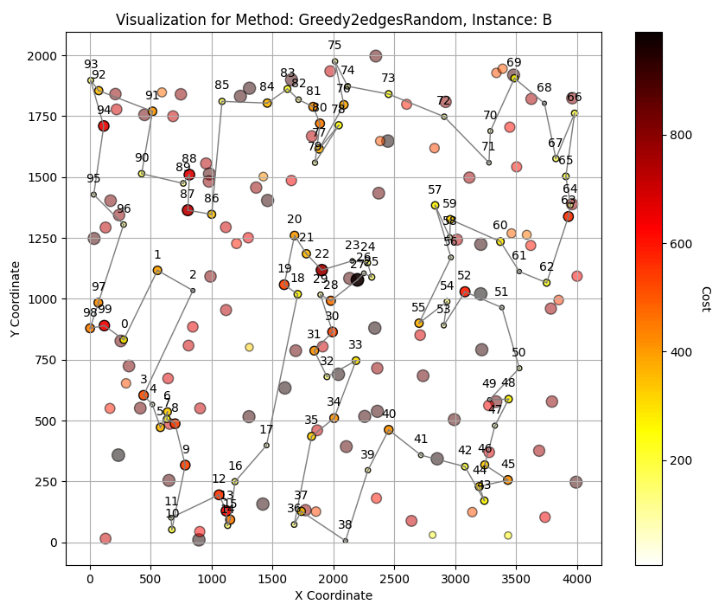


Instance: B

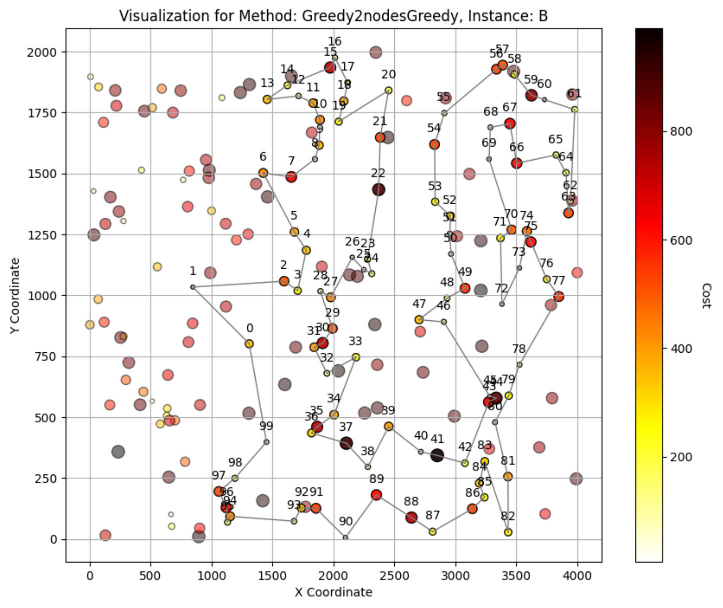
Method: Greedy2edgesGreedy with score=50579



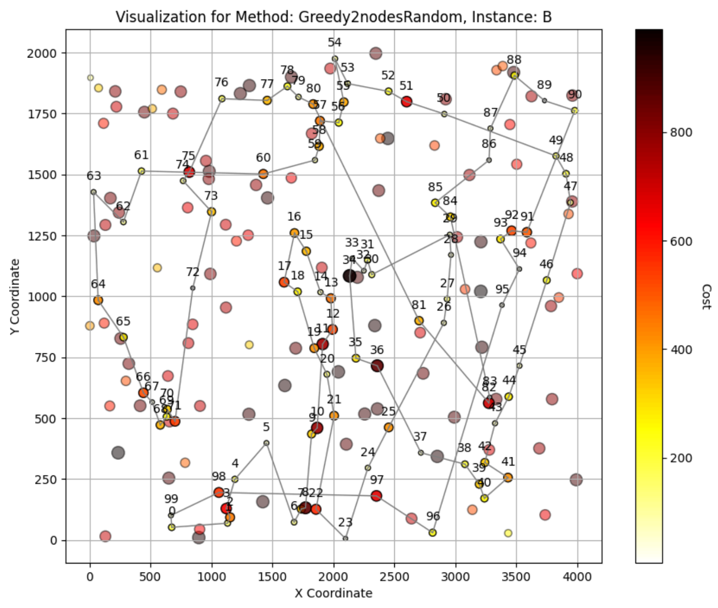
Method: Greedy2edgesRandom with score=46899



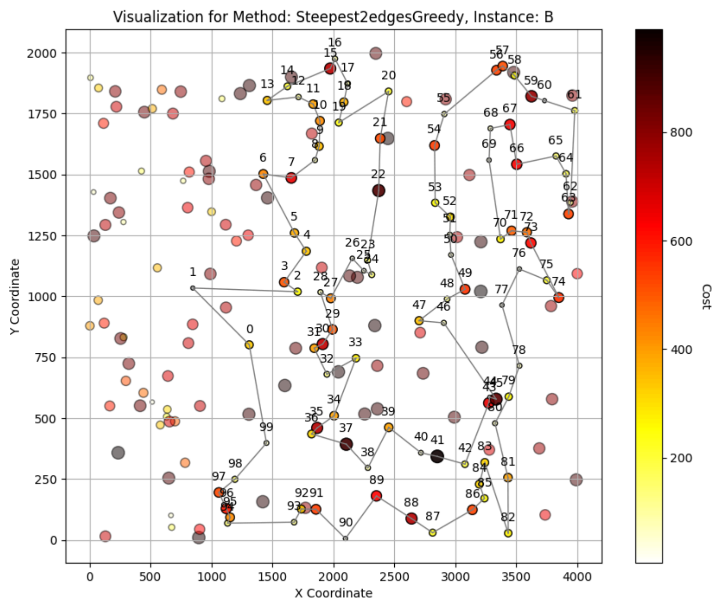
Method: Greedy2nodesGreedy with score=51728



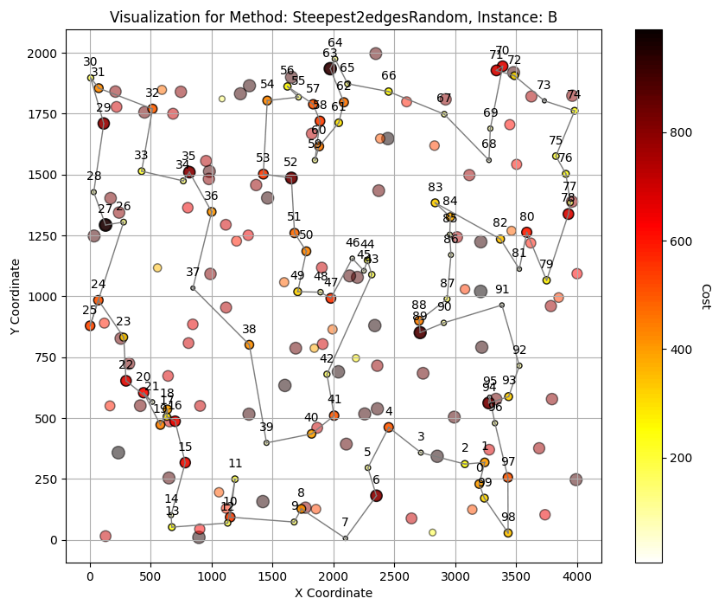
Method: Greedy2nodesRandom with score=53818



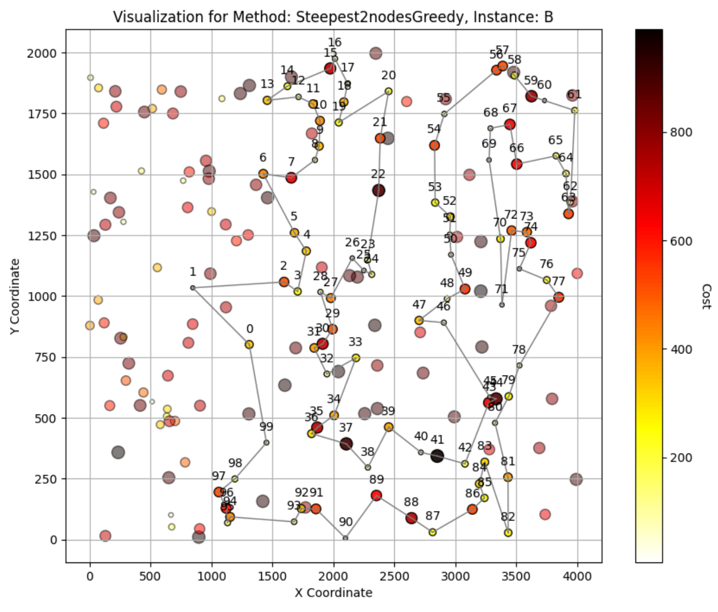
Method: Steepest2edgesGreedy with score=51790



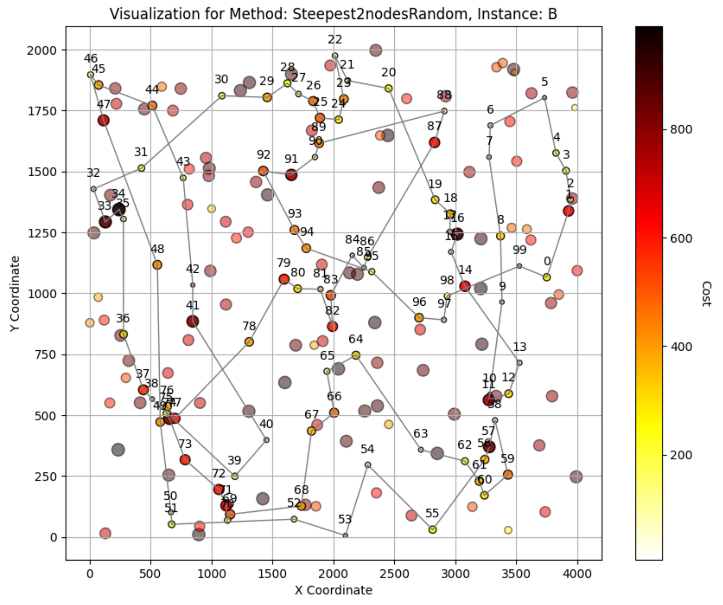
Method: Steepest2edgesRandom with score=46569



Method: Steepest2nodesGreedy with score=51865



Method: Steepest2nodesRandom with score=54443



Best solutions, indices

Instance: A

Method: Greedy2edgesGreedy

Lowest Objective Function Value (f_val): 71612

Solution:

171, 175, 113, 56, 31, 145, 78, 92, 179, 196, 81, 90, 40, 165, 185, 106, 178, 3, 14, 144, 62, 9, 148, 102, 49, 52, 55, 57, 129, 2, 152, 124, 94, 63, 79, 80, 176, 133, 151, 51, 118, 59, 115, 46, 0, 137, 23, 186, 89, 183, 143, 117, 93, 140, 68, 139, 193, 41, 5, 42, 181, 159, 69, 108, 18, 22, 146, 34, 160, 48, 54, 177, 10, 190, 84, 184, 43, 116, 65, 131, 149, 162, 123, 127, 70, 135, 154, 180, 53, 100, 26, 97, 1, 101, 86, 75, 120, 44, 25, 16

Method: Greedy2edgesRandom

Lowest Objective Function Value (f_val): 72146

Solution:

54, 30, 34, 160, 48, 184, 42, 43, 65, 47, 116, 115, 193, 41, 159, 146, 22, 18, 108, 140, 68, 139, 198, 46, 0, 117, 143, 183, 89, 137, 176, 80, 51, 118, 59, 162, 151, 133, 79, 122, 63, 94, 124, 148, 62, 9, 49, 144, 14, 3, 178, 106, 52, 185, 40, 119, 165, 90, 27, 81, 196, 157, 31, 113, 175, 171, 16, 78, 145, 179, 55, 57, 92, 129, 25, 44, 120, 2, 75, 86, 101, 1, 97, 152, 26, 100, 121, 53, 180, 154, 135, 70, 127, 123, 84, 112, 4, 190, 10, 177

Method: Greedy2nodesGreedy

Lowest Objective Function Value (f_val): 71624

Solution:

171, 175, 113, 56, 31, 78, 145, 92, 179, 196, 81, 90, 165, 40, 185, 106, 178, 3, 14, 144, 62, 9, 148, 102, 49, 52, 55, 57, 129, 2, 101, 1, 97, 26, 100, 53, 180, 154, 135, 70, 127, 123, 162, 149, 131, 65, 116, 43, 184, 84, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 41, 193, 139, 68, 140, 93, 117, 143, 183, 89, 186, 23, 137, 0, 46, 115, 59, 118, 51, 151, 133, 176, 80, 79, 63, 94, 124, 152, 86, 75, 120, 44, 25, 16

Method: Greedy2nodesRandom

Lowest Objective Function Value (f_val): 79581

Solution:

196, 40, 185, 57, 129, 124, 94, 122, 63, 121, 26, 100, 53, 70, 127, 112, 84, 59, 137, 23, 186, 89, 183, 143, 0, 117, 108, 18, 69, 139, 68, 46, 198, 193, 159, 22, 41, 115, 118, 51, 176, 80, 97, 1, 101, 75, 86, 180, 154, 135, 162, 133, 151, 65, 116, 43, 42, 181, 34, 160, 54, 177, 184, 35, 131, 149, 123, 79, 152, 2, 120, 44, 16, 171, 175, 113, 31, 179, 106, 178, 52, 55, 145, 78, 92, 148, 9, 62, 49, 164, 7, 21, 144, 14, 138, 165, 39, 27, 90, 81

Method: Steepest2edgesGreedy
Lowest Objective Function Value (f_val): 71865

Solution:

171, 175, 113, 56, 31, 145, 78, 92, 179, 196, 81, 90, 40, 165, 185, 106, 178, 3, 14, 144, 62, 9, 148, 102, 49, 52, 55, 57, 129, 2, 152, 97, 1, 101, 100, 26, 124, 94, 63, 79, 80, 176, 133, 151, 51, 118, 59, 115, 46, 0, 137, 23, 186, 89, 183, 143, 117, 93, 140, 68, 139, 193, 41, 5, 42, 181, 159, 69, 108, 18, 22, 146, 34, 160, 48, 54, 177, 10, 190, 84, 184, 43, 116, 65, 131, 149, 162, 123, 127, 70, 135, 154, 180, 53, 86, 75, 120, 44, 25, 16

Method: Steepest2edgesRandom
Lowest Objective Function Value (f_val): 73122

Solution:

70, 127, 123, 112, 84, 4, 190, 10, 177, 184, 54, 48, 160, 34, 181, 146, 22, 159, 193, 41, 139, 115, 46, 0, 143, 183, 137, 80, 176, 51, 109, 59, 118, 65, 116, 5, 42, 43, 47, 149, 131, 162, 151, 133, 79, 63, 94, 124, 148, 62, 9, 144, 102, 49, 14, 138, 106, 178, 52, 55, 185, 119, 40, 165, 39, 95, 7, 164, 90, 196, 81, 157, 56, 113, 171, 175, 16, 31, 145, 78, 179, 57, 92, 129, 25, 44, 120, 2, 152, 97, 1, 101, 75, 86, 100, 26, 53, 180, 154, 135

Method: Steepest2nodesGreedy
Lowest Objective Function Value (f_val): 71624

Solution:

171, 175, 113, 56, 31, 78, 145, 92, 179, 196, 81, 90, 165, 40, 185, 106, 178, 3, 14, 144, 62, 9, 148, 102, 49, 52, 55, 57, 129, 2, 101, 1, 97, 26, 100, 53, 180, 154, 135, 70, 127, 123, 162, 149, 131, 65, 116, 43, 184, 84, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 41, 193, 139, 68, 140, 93, 117, 143, 183, 89, 186, 23, 137, 0, 46, 115, 59, 118, 51, 151, 133, 176, 80, 79, 63, 94, 124, 152, 86, 75, 120, 44, 25, 16

Method: Steepest2nodesRandom
Lowest Objective Function Value (f_val): 81006

Solution:

63, 182, 136, 154, 135, 70, 127, 123, 112, 4, 177, 54, 193, 41, 42, 43, 149, 131, 184, 160, 34, 146, 22, 18, 0, 153, 89, 183, 143, 117, 68, 46, 139, 115, 118, 51, 176, 124, 167, 57, 179, 196, 81, 40, 106, 49, 102, 144, 14, 178, 52, 55, 92, 129, 152, 100, 133, 162, 65, 116, 59, 151, 75, 120, 44, 25, 78, 145, 157, 31, 16, 171, 175, 113, 90, 165, 185, 82, 2, 1, 97, 26, 101, 86, 158, 180, 53, 121, 94, 148, 37, 9, 62, 15, 186, 23, 137, 80, 79, 122

Instance: B

Method: Greedy2edgesGreedy
Lowest Objective Function Value (f_val): 50579

Solution:

190, 80, 175, 78, 5, 177, 36, 61, 91, 141, 97, 146, 187, 165, 127, 89, 103, 137, 114, 113, 194, 166, 172, 179, 99, 22, 185, 86, 95, 130, 183, 140, 199, 9, 148, 47, 66, 94, 60, 20, 59, 28, 149, 4, 152, 170, 34, 55, 18, 62, 128, 124, 143, 106, 88, 176, 180, 163, 186, 153, 81, 77, 58, 21, 87, 82, 111, 8, 104, 56, 144, 33, 160, 29, 0, 35, 109, 69, 189, 155, 145, 15, 3, 70, 161, 188, 6, 169, 132, 13, 195, 168, 43, 11, 139, 138, 25, 121, 117, 31

Method: Greedy2edgesRandom
Lowest Objective Function Value (f_val): 46899

Solution:

1, 131, 121, 198, 117, 193, 54, 31, 73, 136, 80, 190, 45, 142, 175, 78, 5, 177, 138, 182, 139, 11, 49, 29, 109, 35, 0, 12, 160, 33, 144, 104, 8, 111, 82, 21, 36, 61, 141, 77, 81, 153, 163, 127, 89, 114, 103, 113, 194, 176, 166, 86, 128, 106, 124, 143, 62, 34, 18, 55, 95, 185, 179, 66, 94, 47, 60, 148, 20, 28, 140, 183, 152, 155, 3, 70, 15, 195, 145, 168, 13, 132, 169, 188, 6, 147, 51, 125, 191, 90, 122, 133, 107, 40, 100, 63, 135, 38, 27, 16

Method: Greedy2nodesGreedy
Lowest Objective Function Value (f_val): 51728

Solution:

25, 121, 182, 138, 11, 139, 134, 43, 168, 195, 13, 132, 169, 6, 188, 161, 70, 3, 15, 145, 155, 189, 69, 109, 35, 0, 29, 160, 33, 144, 56, 104, 8, 111, 82, 87, 21, 58, 77, 81, 153, 186, 163, 180, 88, 176, 106, 143, 124, 128, 62, 18, 55, 34, 170, 152, 4, 149, 28, 59, 20, 60, 94, 66, 47, 148, 9, 199, 140, 183, 130, 95, 86, 185, 99, 22, 179, 172, 166, 194, 113, 114, 137, 103, 89, 127, 165, 187, 146, 97, 141, 91, 61, 36, 78, 175, 142, 45, 5, 177

Method: Greedy2nodesRandom
Lowest Objective Function Value (f_val): 53818

Solution:

80, 175, 78, 142, 5, 177, 36, 61, 79, 21, 87, 56, 144, 160, 33, 11, 139, 182, 138, 104, 8, 82, 91, 141, 77, 81, 106, 124, 62, 18, 35, 109, 0, 29, 39, 111, 41, 153, 163, 89, 127, 114, 103, 113, 194, 166, 179, 94, 47, 148, 152, 184, 155, 3, 70, 15, 145, 13, 195, 168, 134, 122, 135, 63, 38, 1, 198, 117, 193, 31, 54, 73, 121, 51, 90, 191, 147, 6, 188, 169, 132, 143, 180, 176, 55, 34, 183, 140, 28, 20, 60, 99, 130, 95, 185, 86, 187, 97, 45, 190

Method: Steepest2edgesGreedy
Lowest Objective Function Value (f_val): 51790

Solution:

25, 121, 138, 182, 11, 139, 134, 43, 168, 195, 13, 132, 169, 6, 188, 161, 70, 3, 15, 145, 155, 189, 69, 109, 35, 0, 29, 160, 33, 144, 56, 104, 8, 111, 82, 87, 21, 58, 77, 81, 153, 186, 163, 180, 176, 88, 106, 143, 124, 128, 62, 18, 55, 34, 170, 152, 4, 149, 28, 59, 20, 60, 94, 66, 47, 148, 9, 199, 140, 183, 95, 130, 99, 22, 172, 179, 185, 86, 166, 194, 113, 114, 137, 103, 89, 127, 165, 187, 146, 97, 141, 91, 61, 36, 175, 78, 142, 45, 5, 177

Method: Steepest2edgesRandom
Lowest Objective Function Value (f_val): 46569

Solution:

89, 103, 163, 153, 81, 77, 97, 141, 61, 36, 78, 5, 175, 80, 190, 136, 73, 31, 54, 193, 198, 117, 156, 1, 38, 27, 135, 102, 63, 100, 40, 107, 133, 122, 90, 191, 51, 121, 25, 177, 21, 82, 8, 35, 109, 0, 29, 160, 33, 138, 11, 139, 43, 134, 6, 169, 188, 132, 13, 168, 195, 145, 15, 161, 70, 3, 155, 152, 183, 140, 149, 4, 28, 20, 60, 148, 47, 94, 66, 179, 99, 185, 95, 34, 55, 18, 62, 124, 143, 159, 106, 86, 166, 194, 180, 176, 113, 114, 137, 127

Method: Steepest2nodesGreedy
Lowest Objective Function Value (f_val): 51865

Solution:

25, 121, 182, 138, 11, 139, 134, 43, 168, 195, 13, 132, 169, 6, 188, 161, 70, 3, 15, 145, 155, 189, 69, 109, 35, 0, 29, 160, 33, 144, 56, 104, 8, 111, 82, 87, 21, 58, 77, 81, 153, 186, 163, 180, 88, 176, 106, 143, 124, 128, 62, 18, 55, 34, 170, 152, 4, 149, 28, 59, 20, 60, 94, 66, 47, 148, 9, 199, 140, 183, 95, 86, 130, 99, 22, 185, 179, 172, 166, 194, 113, 114, 137, 103, 89, 127, 165, 187, 146, 97, 141, 91, 61, 36, 78, 175, 142, 45, 5, 177

Method: Steepest2nodesRandom
Lowest Objective Function Value (f_val): 54443

Solution:

179, 66, 94, 47, 148, 20, 140, 183, 95, 86, 176, 180, 194, 166, 128, 62, 83, 18, 55, 34, 155, 3, 70, 15, 145, 13, 132, 169, 188, 6, 147, 122, 63, 102, 32, 135, 1, 198, 117, 5, 177, 112, 121, 90, 133, 107, 40, 100, 131, 193, 190, 80, 36, 141, 77, 187, 103, 26, 113, 114, 127, 89, 163, 153, 111, 8, 82, 21, 61, 78, 175, 142, 45, 136, 164, 31, 54, 73, 25, 182, 138, 33, 144, 160, 29, 0, 109, 170, 152, 195, 168, 43, 134, 139, 11, 35, 143, 106, 124, 185

Conclusions

We were to implement both steepest and greedy version of local search. As the type of neighborhood two kinds of moves were being used: 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. For intra-route moves two options were used: two-nodes exchange, two-edges exchange. Moreover, we used two types of starting solutions: random & greedy with 2regret weighted sum. Totalling in 8 new methods this week. The top-performing method so far, based on the objective function value, was the Steepest2nodes with 2regret heuristic for instance A, and for instance B the same algorithm but with random initialization.