Assignment 4: The use of candidate moves in local search

Authors: Kiril Andrukh, 162069; Uladzislau Lukashevich, 155671.
Source code: link

Description of the problem

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.

Local Search

Pseudocode:

```
Initialize cost of the initial solution
Set the solution as the initial solution
Identify selected nodes and non-selected nodes
Initialize candidate edges for each node (k nearest neighbors based on distance + cost)

Loop until no improvement can be found:
    Search for intra-route neighbors:
        For each node i in the route:
            For each candidate neighbour of the node i:
                Calculate potential improvement (delta)
                If delta < 0:
                    Add (i, j, delta, "edge") to intra-route neighbors

    Search for inter-route neighbors:
        For each node i in solution:
            For each candidate neighbour of the node i in non-selected nodes:
                Calculate potential improvement (delta)
                If delta < 0:
                    Add (i, candidate, delta, "inter") to inter-route neighbors

    Combine all neighbors into a single list
```

```
    If no improving moves exist:
        Exit the loop

    Select the move with the steepest improvement

    Apply the selected move:
        Update the solution, selected nodes, and non-selected nodes
        Update the cost of the solution

Return the final solution
```

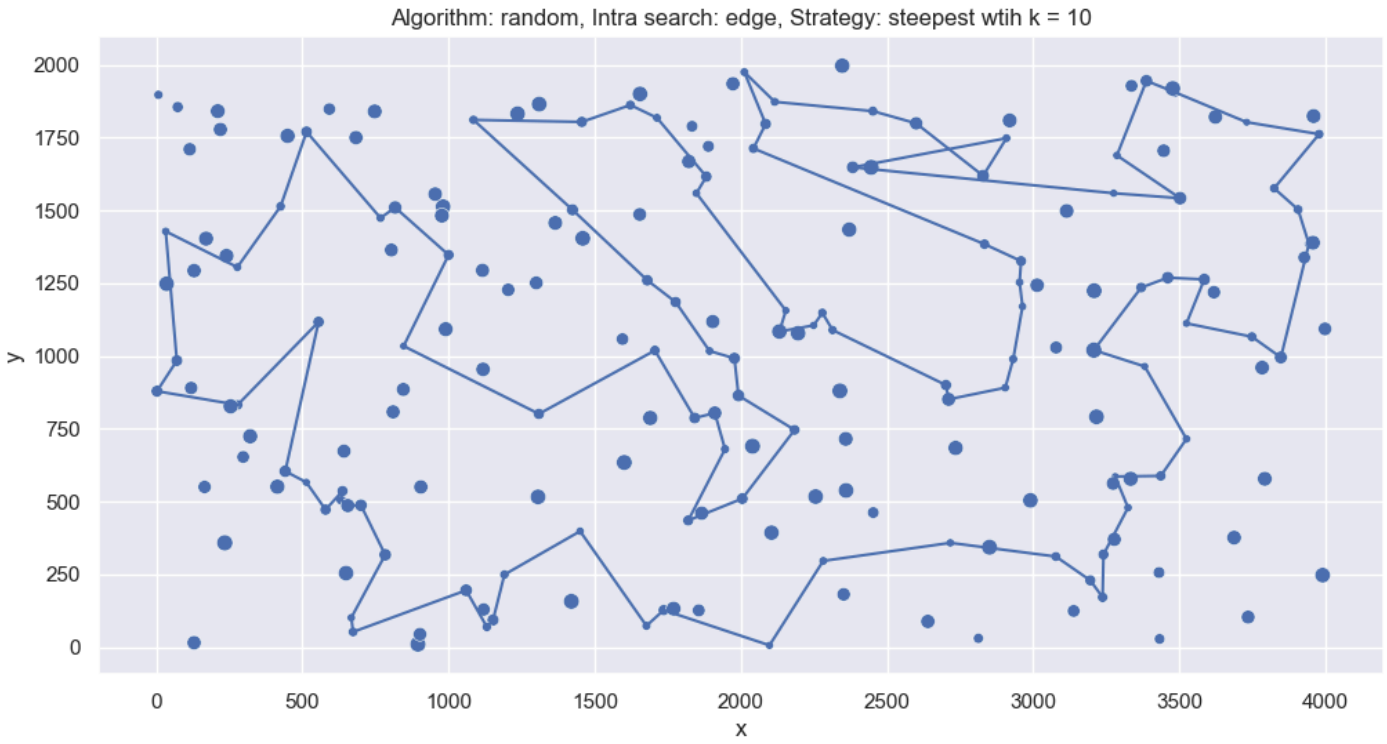
With k = 10

Dataset A:

Best solution: [135, 70, 158, 180, 173, 151, 79, 80, 176, 66, 60, 118, 116, 162, 161, 156, 4, 190, 177, 104, 184, 166, 28, 43, 42, 5, 96, 139, 191, 142, 193, 159, 192, 34, 103, 195, 22, 20, 108, 36, 93, 170, 143, 153, 183, 89, 83, 64, 15, 23, 137, 148, 37, 102, 144, 14, 21, 95, 164, 71, 27, 165, 106, 178, 3, 138, 32, 49, 167, 111, 87, 152, 97, 75, 2, 44, 16, 171, 175, 113, 56, 31, 157, 81, 187, 169, 196, 145, 78, 82, 92, 57, 52, 94, 63, 182, 189, 99, 121, 24]

Dataset B:

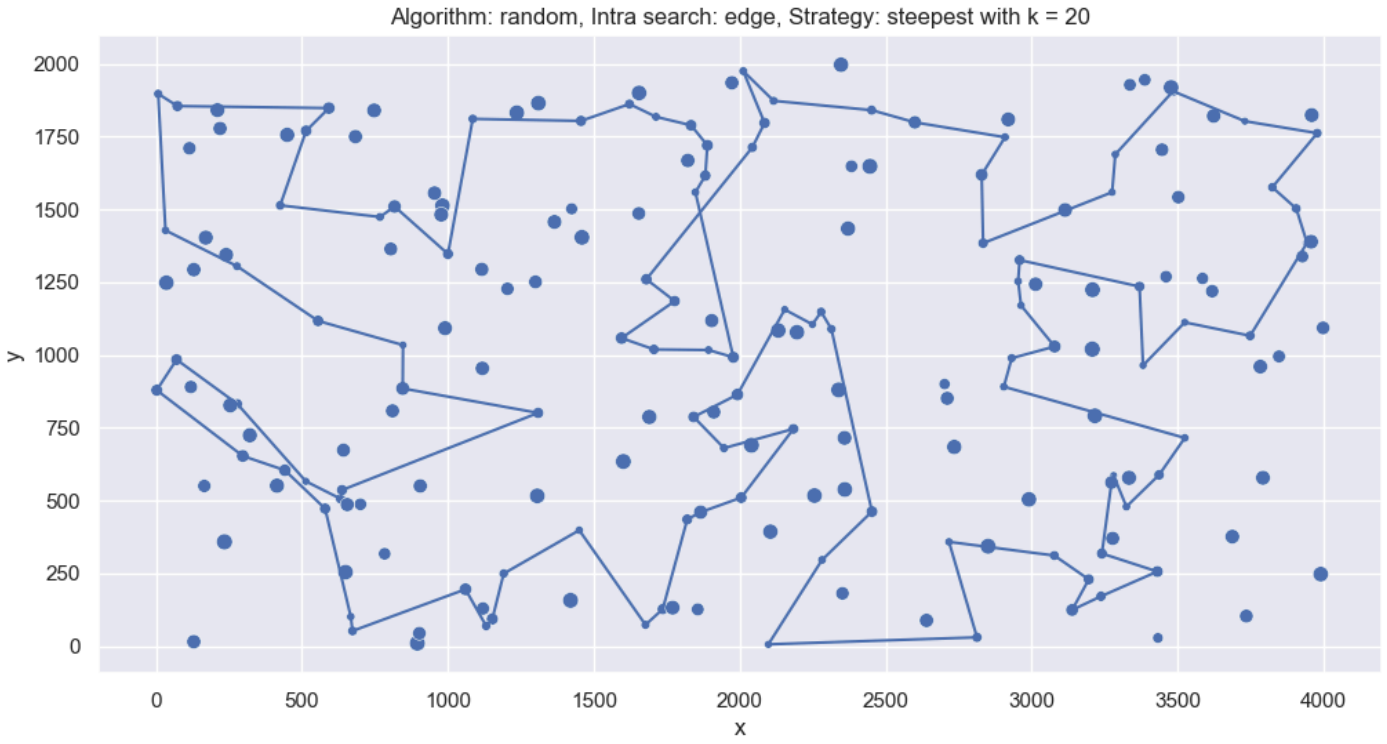
Best solution: [80, 73, 164, 54, 31, 193, 30, 42, 198, 117, 151, 123, 177, 171, 157, 56, 144, 35, 109, 29, 11, 139, 182, 74, 51, 158, 25, 19, 1, 197, 92, 96, 17, 135, 131, 90, 191, 71, 115, 147, 134, 6, 188, 169, 126, 13, 161, 70, 3, 84, 167, 152, 53, 4, 28, 174, 34, 18, 83, 62, 128, 124, 159, 81, 163, 26, 113, 180, 176, 88, 194, 166, 86, 181, 95, 130, 148, 20, 23, 60, 154, 66, 179, 52, 48, 75, 127, 165, 97, 77, 58, 41, 111, 8, 21, 79, 61, 78, 45, 46]



With k = 20

Dataset A:

Best solution: [164, 95, 107, 90, 165, 119, 40, 185, 196, 81, 98, 31, 13, 113, 175, 171, 16, 78, 145, 179, 91, 82, 120, 2, 75, 1, 26, 53, 158, 180, 121, 99, 189, 97, 152, 125, 167, 128, 57, 55, 52, 106, 138, 14, 144, 62, 9, 37, 137, 176, 80, 79, 151, 162, 161, 6, 127, 123, 156, 126, 4, 24, 149, 166, 28, 30, 104, 147, 181, 5, 96, 65, 197, 72, 109, 51, 141, 60, 46, 110, 193, 159, 195, 103, 22, 199, 69, 163, 67, 36, 140, 170, 143, 183, 89, 23, 186, 114, 21, 7]



Dataset B:

Best solution: [169, 126, 13, 145, 15, 70, 3, 189, 155, 184, 170, 34, 35, 11, 139, 138, 33, 144, 56, 104, 8, 82, 50, 14, 81, 153, 106, 128, 62, 55, 181, 95, 185, 99, 9, 183, 149, 20, 23, 60, 148, 47, 94, 66, 172, 52, 48, 194, 113, 103, 137, 127, 163, 187, 146, 97, 77, 141, 21, 79, 61, 7, 5, 78, 162, 31, 193, 198, 196, 42, 156, 24, 1, 197, 27, 38, 63, 100, 72, 17, 10, 115, 178, 122, 32, 135, 131, 112, 19, 25, 116, 90, 71, 192, 134, 85, 98, 74, 43, 65]

With k = 50

Dataset A: [136, 180, 154, 6, 70, 135, 133, 79, 63, 122, 176, 66, 141, 51, 109, 59, 118, 60, 46, 139, 115, 43, 47, 123, 127, 156, 112, 4, 84, 166, 184, 190, 10, 177, 54, 48, 34, 160, 192, 42, 5, 96, 193, 22, 199, 108, 36, 140, 93, 117, 153, 76, 186, 114, 15, 73, 132, 21, 7, 164, 58, 71, 95, 165, 138, 144, 9, 148, 167, 111, 124, 152, 172, 55, 3, 106, 185, 40, 187, 169, 196, 157, 188, 85, 113, 175, 13, 145, 92, 129, 82, 25, 120, 87, 97, 1, 101, 75, 86, 182]

Dataset B: [64, 128, 62, 143, 119, 14, 50, 68, 111, 35, 109, 12, 39, 144, 160, 33, 49, 138, 11, 168, 195, 69, 189, 167, 155, 84, 15, 145, 13, 169, 188, 118, 125, 71, 10, 133, 44, 107, 100, 63, 96, 135, 38, 121, 158, 151, 198, 117, 54, 31, 164, 73, 190, 80, 46, 45, 142, 175, 78, 7, 123, 25, 157, 36, 91, 82, 58, 77, 97, 146, 153, 127, 137, 75, 114, 103, 26, 113, 88, 194, 166, 86, 185, 179, 57, 66, 148, 9, 140, 28, 4, 53, 152, 170, 34, 18, 55, 183, 95, 110]

Final tables:

Function performance

Method	Dataset A	Dataset B
Random generation, edge, steepest	73855.835(70939-77610)	48296.625(45319-50992)
Random, edge, steepest with 10 candidates	84844.105(78476-97467)	53100.91(49045-60284)
Random, edge, steepest with 20 candidates	85070.815(78269-94962)	52889.535(48412-60756)
Random, edge, steepest with 50 candidates	85004.05(78621-93034)	53010.765(48338-60043)

Average running time

Method	Dataset A	Dataset B
Random generation, edge, steepest	3.3 s	3.17 s
Random, edge, steepest with 10 candidates	1.25 s	1.54 s
Random, edge, steepest with 20 candidates	1.22 s	1.36 s
Random, edge, steepest with 50 candidates	1.16 s	1.23 s

Conclusion:

Looking at the results we can say that using candidate moves decreases the running time of the algorithm: there is almost a 3 times improvement to the algorithm. However, with use of candidate moves, the objective function becomes a bit worse.