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 with Candidates Moves

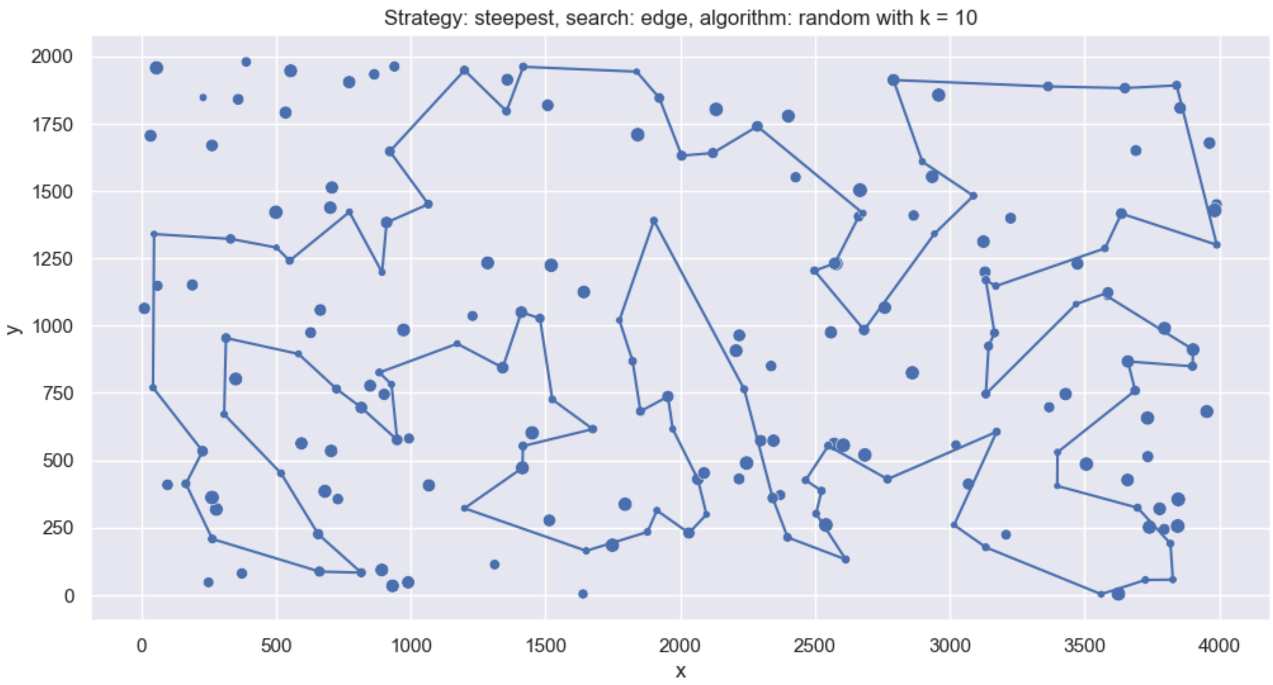
Pseudocode:

```
Initialize cost of the initial solution
Set the solution as the initial solution
Identify selected nodes and non-selected nodes

Loop until no improvement can be found:
    Initialize list of all possible improving moves from each node in cycle to its k closest vertices (distance + cost wise)

    For candidate in list:
        If candidate is within a cycle:
            Intra edges exchange
        Else if candidate is not within a cycle:
            Inter nodes exchange

    If there are no improving neighbors:
        Exit the loop

    Choose the neighbor with the steepest improvement

    Update solution based on the best neighbor
    Update cost by adding the improvement of the best neighbor

Return the final solution
```

With k = 10

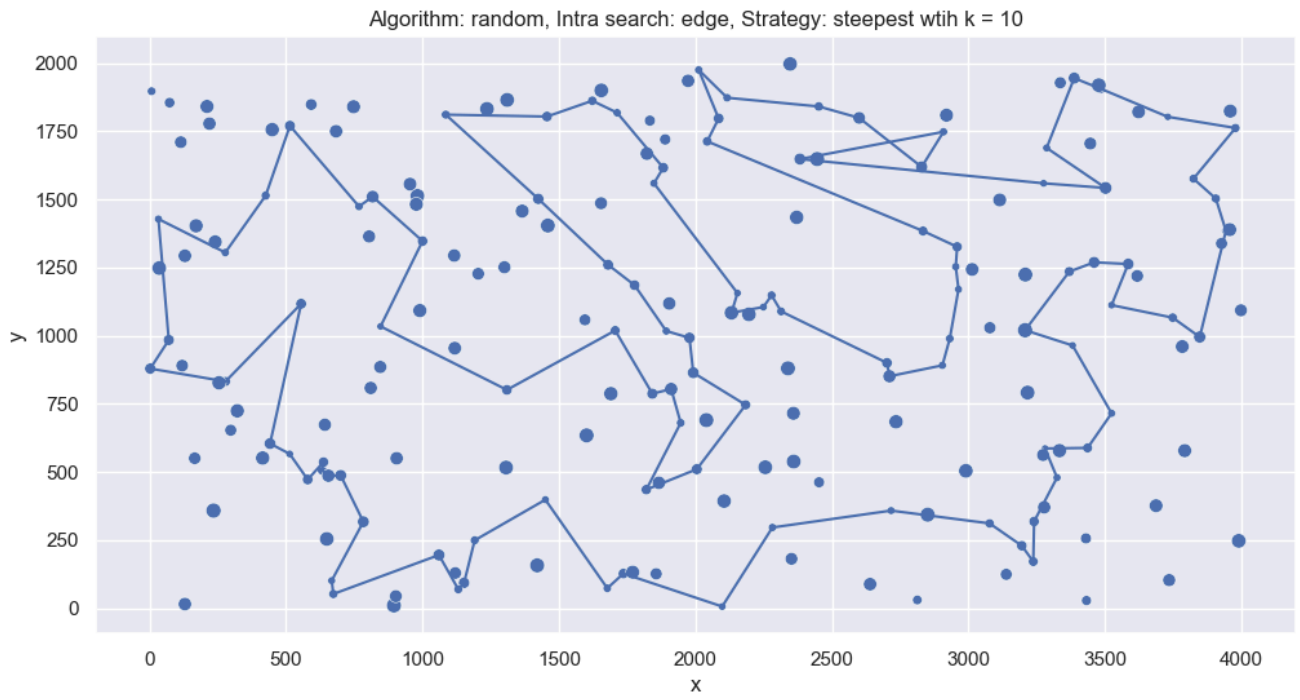
Dataset A:

Best solution: [62, 9, 37, 148, 167, 49, 14, 144, 73, 21, 7, 164, 90, 39, 165, 106, 178, 52, 55, 57, 185, 119, 40, 187, 81, 169, 196, 145, 78, 31, 113, 175, 171, 16, 44, 120, 92, 2, 152, 97, 1, 101, 75, 86, 100, 94, 137, 176, 80, 79, 122, 63, 136, 53, 158, 180, 154, 135, 123, 161, 162, 133, 151, 51, 109, 72, 59, 116, 65, 131, 77, 43, 42, 181, 160, 184, 84, 112, 4, 177, 54, 48, 34, 22, 159, 193, 41, 139, 115, 198, 46, 68, 117, 0, 143, 183, 89, 23, 186, 114]


Strategy: steepest, search: edge, algorithm: random with k = 10

Dataset B:

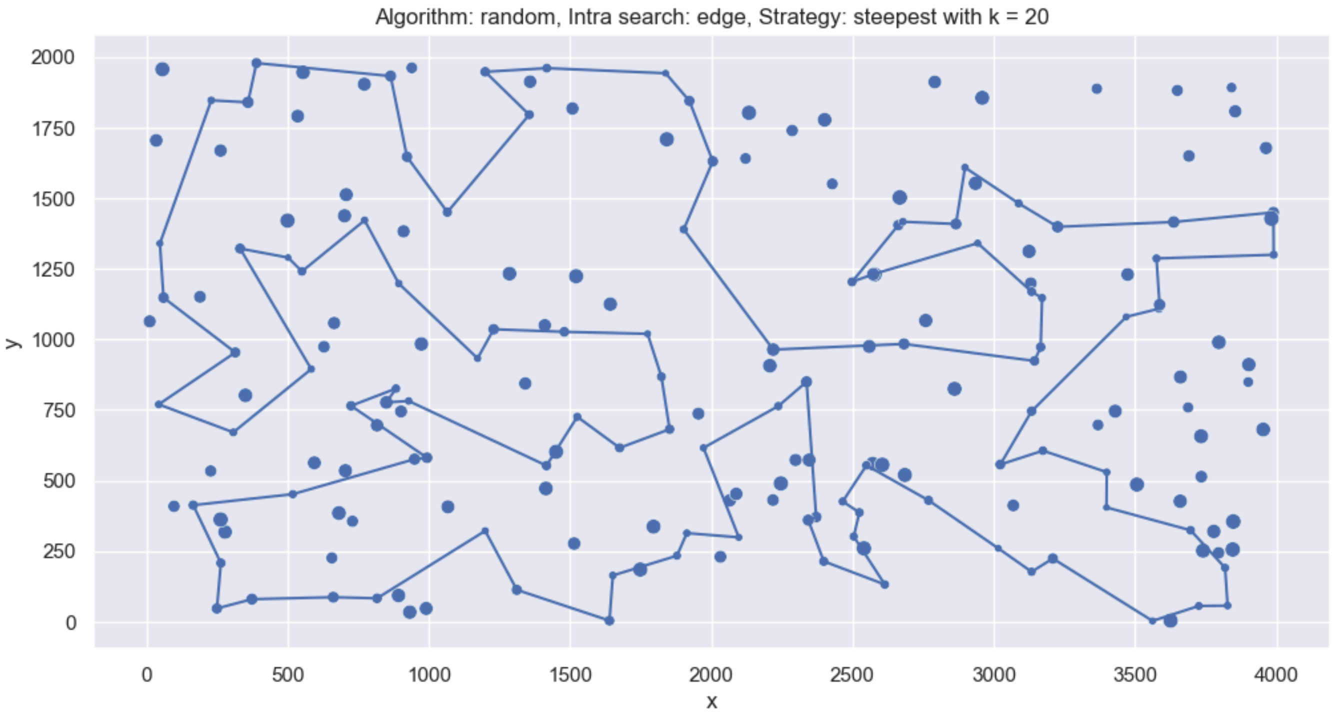
Best solution: [152, 189, 183, 9, 140, 149, 28, 20, 60, 148, 47, 94, 172, 179, 185, 99, 130, 95, 110, 86, 166, 194, 176, 113, 103, 127, 89, 163, 153, 77, 141, 61, 36, 177, 5, 78, 175, 45, 80, 190, 136, 73, 164, 31, 54, 193, 117, 198, 131, 1, 27, 38, 63, 135, 122, 133, 90, 191, 51, 121, 25, 138, 104, 56, 8, 21, 82, 111, 144, 160, 33, 11, 139, 134, 147, 6, 188, 169, 195, 168, 29, 39, 0, 109, 35, 143, 159, 106, 124, 62, 18, 55, 34, 145, 15, 70, 3, 155, 184, 170]



With k = 20

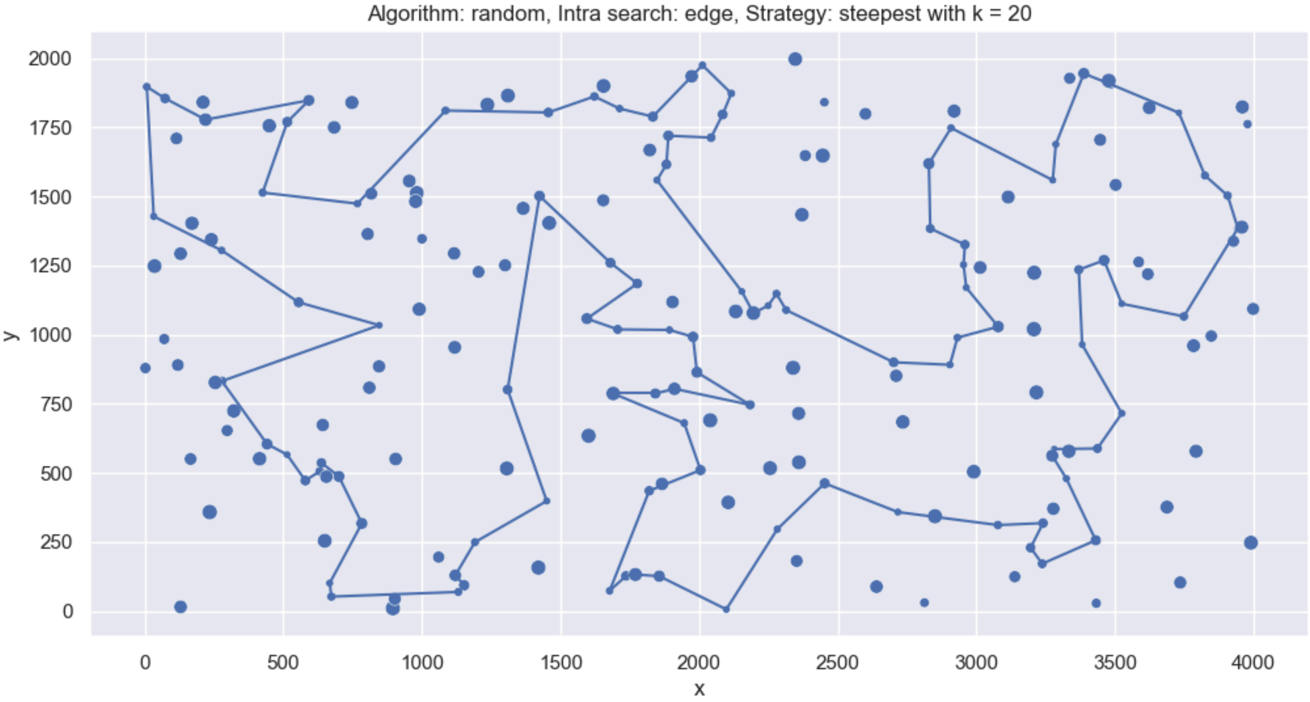
Dataset A:

Best solution: [55, 52, 106, 178, 49, 37, 148, 9, 62, 102, 144, 14, 138, 39, 27, 90, 165, 119, 40, 185, 57, 129, 92, 145, 78, 31, 113, 175, 171, 16, 25, 44, 120, 2, 152, 97, 1, 101, 75, 86, 100, 26, 124, 94, 63, 53, 180, 154, 135, 70, 127, 123, 112, 4, 190, 10, 177, 54, 184, 131, 149, 43, 116, 105, 65, 162, 151, 133, 79, 80, 176, 51, 118, 59, 115, 139, 41, 193, 159, 42, 160, 34, 181, 146, 22, 18, 69, 108, 140, 68, 46, 0, 117, 143, 183, 89, 23, 137, 12, 167]



Dataset B:

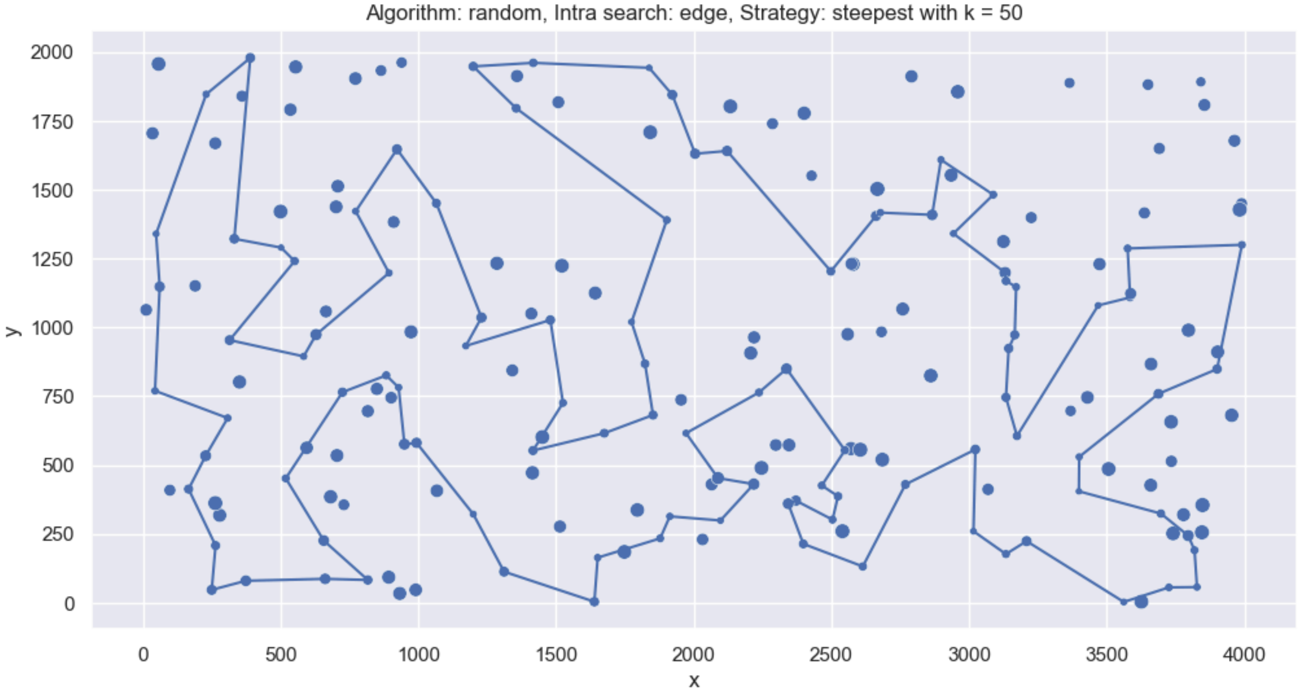
Best solution: [107, 40, 63, 135, 131, 121, 1, 198, 117, 193, 31, 54, 73, 136, 190, 80, 175, 78, 142, 5, 177, 25, 134, 139, 11, 182, 138, 33, 160, 144, 111, 56, 104, 157, 8, 82, 21, 36, 61, 79, 91, 141, 77, 81, 153, 163, 103, 89, 127, 114, 113, 180, 176, 194, 166, 86, 95, 130, 185, 179, 94, 47, 148, 20, 28, 149, 140, 183, 152, 170, 34, 55, 18, 62, 128, 124, 106, 143, 35, 109, 0, 12, 29, 168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6, 147, 90, 122, 133, 10, 72]



With k = 50

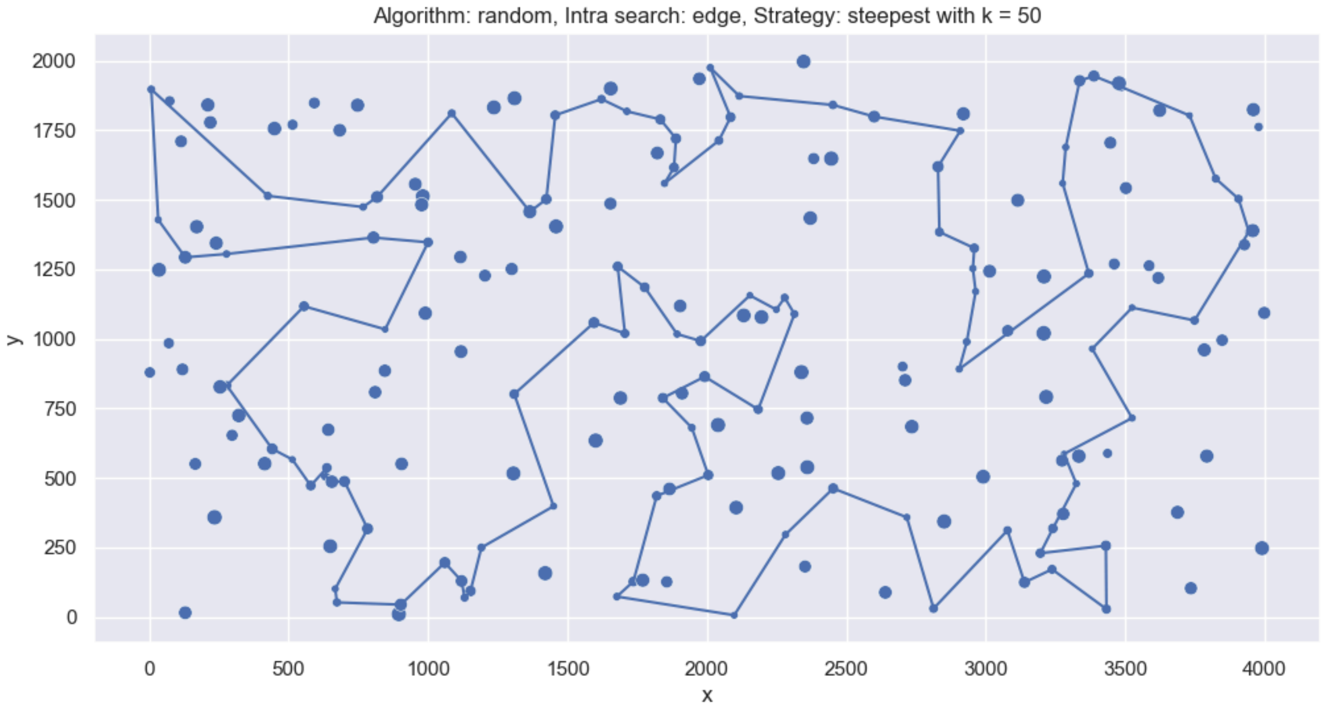
Dataset A:

Best solution: [133, 79, 80, 176, 137, 0, 117, 143, 183, 89, 23, 186, 148, 9, 62, 102, 144, 14, 49, 3, 178, 106, 52, 55, 57, 92, 185, 40, 165, 90, 81, 196, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 129, 2, 75, 86, 100, 26, 101, 1, 97, 152, 124, 94, 63, 182, 121, 53, 180, 154, 135, 70, 127, 123, 149, 131, 65, 116, 43, 184, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 108, 159, 193, 41, 181, 42, 5, 115, 139, 68, 46, 118, 59, 51, 151, 162]



Dataset B:

Best solution: [162, 45, 142, 175, 78, 5, 177, 25, 182, 138, 139, 11, 33, 160, 29, 0, 109, 35, 111, 144, 104, 8, 82, 21, 61, 36, 141, 77, 81, 153, 187, 163, 165, 127, 137, 114, 89, 103, 113, 176, 166, 86, 185, 179, 94, 47, 148, 20, 28, 149, 4, 140, 183, 95, 106, 124, 62, 18, 55, 34, 170, 152, 184, 155, 3, 70, 15, 145, 168, 195, 13, 132, 169, 188, 6, 134, 85, 147, 191, 90, 122, 40, 63, 102, 135, 125, 51, 121, 131, 1, 198, 117, 193, 54, 31, 164, 73, 136, 190, 80]



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	84871.46(78476-97467)	53099.35(49045-60284)
Random, edge, steepest with 20 candidates	77894.29(73484-84130)	49239.81(46476-52665)
Random, edge, steepest with 50 candidates	75074.33(71128-81175)	49092.33(46440-52405)

Average running time

Method	Dataset A	Dataset B
Random generation, edge, steepest	3.3 s	3.17 s
Random, edge, steepest with 10 candidates	0.79 s	0.91 s
Random, edge, steepest with 20 candidates	1.22 s	1.21 s

Random, edge, steepest with 50 candidates	2.32 s	2.45 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.