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:

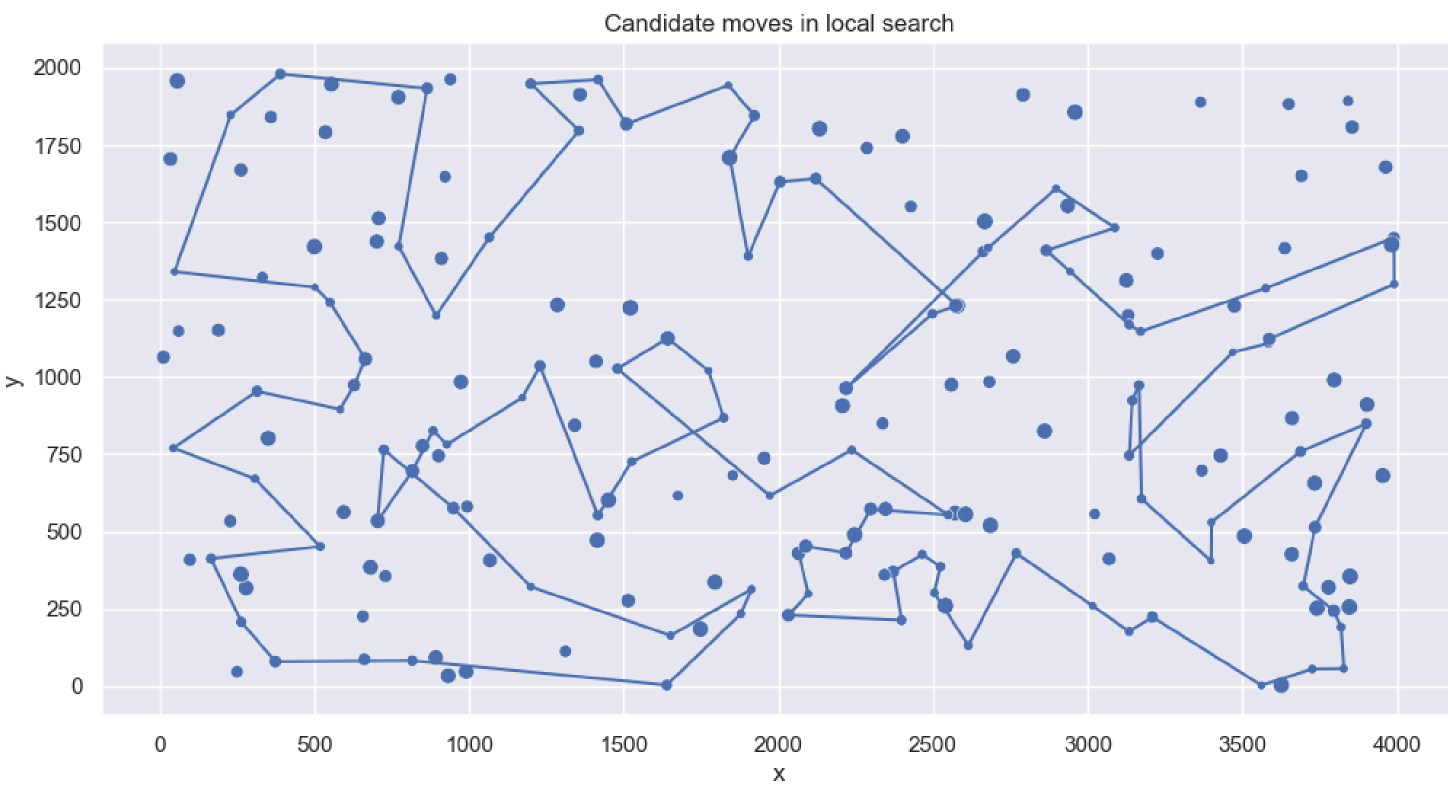
```
    Initialize cost of the initial solution
Set the solution as the initial solution
Identify selected nodes and non-selected nodes
Initialize candidate edges for each node (k nearest neighbors based on distance + cost)

Loop until no improvement can be found:
    Search for intra-route neighbors:
        For each pair of nodes (i,j) in solution:
            Calculate potential improvement (delta)
            If delta < 0 AND (node[i] is in candidates of node[j] OR node[j] is in
candidates of node[i]):
                Add (i,j,delta,"edge") to intra-route neighbors

    Search for inter-route neighbors:
        For each node i in solution and vacant_node in non-selected nodes:
            Calculate potential improvement (delta)
            If delta < 0 AND (vacant_node is in candidates of node[i] OR node[i] is in
candidates of vacant_node):
                Add (i,vacant_node,delta,"inter") to inter-route neighbors

    Combine intra-route and inter-route neighbors into all_neighbors
```

```
    If there are no improving neighbors:
        Exit the loop

    If strategy is "greedy":
        Shuffle neighbors and select the first improving neighbor
    Else If strategy is "steepest":
        Choose the neighbor with the steepest improvement

    Update solution, selected nodes, and non-selected nodes based on the best neighbor
    Update cost by adding the improvement of the best neighbor

Return the final solution
```

With k = 10

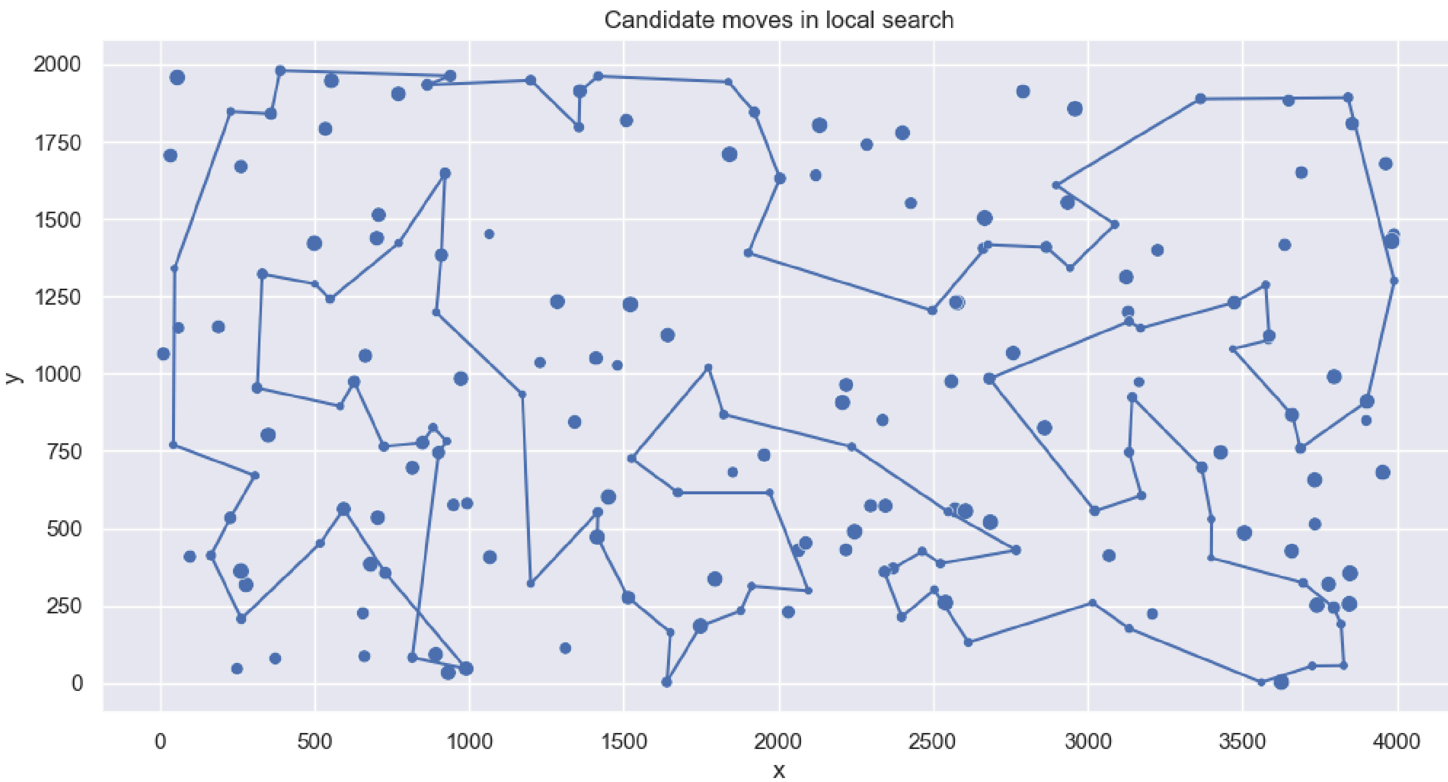
Dataset A:

Best solution: [54, 177, 190, 112, 70, 154, 180, 135, 123, 131, 43, 166, 77, 116, 65, 59, 118, 162, 151, 80, 176, 66, 51, 63, 94, 152, 189, 121, 182, 136, 53, 158, 86, 26, 97, 1, 101, 75, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 157, 81, 196, 145, 78, 92, 52, 55, 57, 185, 40, 119, 90, 27, 165, 106, 178, 49, 102, 14, 144, 62, 9, 12, 148, 33, 186, 23, 137, 76, 89, 183, 153, 143, 117, 0, 46, 115, 139, 140, 108, 18, 22, 193, 41, 96, 5, 42, 181, 34, 160, 184]



Dataset B:

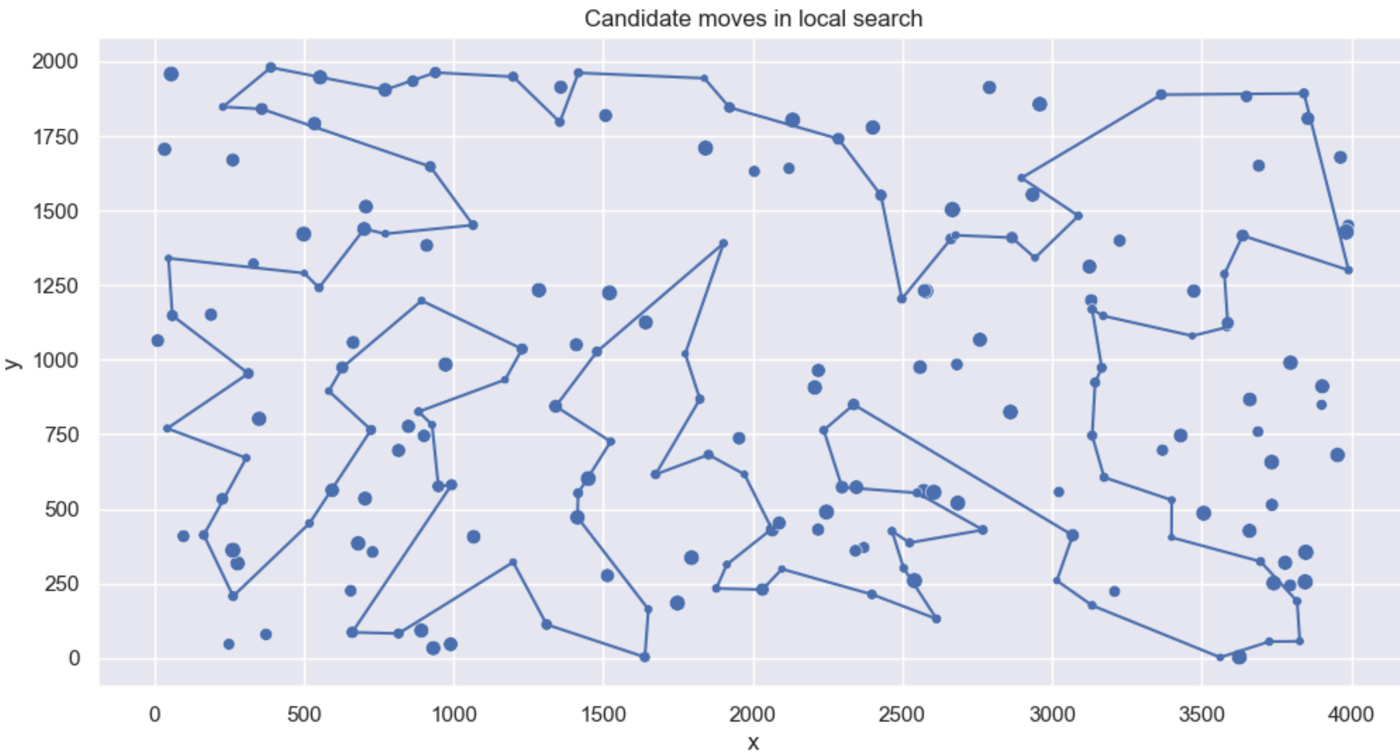
Best solution: [164, 21, 144, 14, 49, 102, 62, 148, 137, 23, 89, 183, 143, 170, 0, 117, 140, 93, 108, 69, 18, 22, 34, 160, 48, 54, 177, 184, 28, 35, 29, 112, 47, 65, 116, 105, 43, 5, 42, 181, 159, 193, 41, 139, 68, 198, 115, 59, 123, 162, 161, 194, 135, 70, 6, 154, 180, 53, 63, 133, 151, 176, 80, 94, 152, 2, 1, 97, 26, 100, 86, 101, 75, 120, 44, 16, 171, 175, 113, 56, 31, 78, 145, 179, 55, 57, 92, 129, 167, 178, 106, 8, 165, 119, 40, 185, 169, 196, 187, 90]



With k = 20

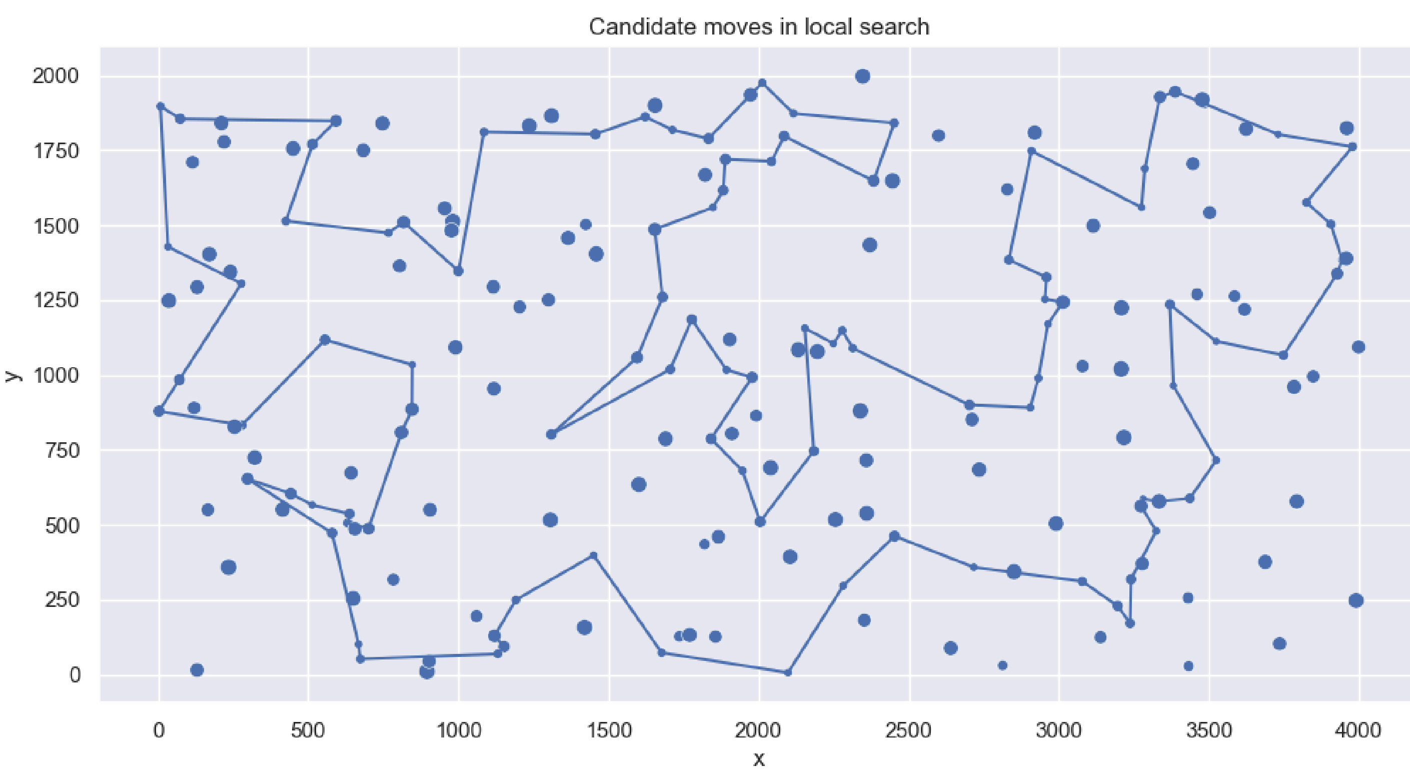
Dataset A:

Best solution:[72, 151, 162, 161, 135, 70, 127, 123, 112, 4, 149, 131, 65, 116, 59, 118, 115, 5, 42, 43, 184, 177, 54, 160, 34, 181, 146, 22, 193, 41, 191, 139, 46, 68, 69, 18, 108, 36, 93, 117, 0, 143, 183, 89, 114, 15, 148, 9, 62, 102, 49, 14, 144, 21, 164, 90, 39, 165, 119, 40, 185, 106, 178, 52, 55, 57, 92, 145, 78, 31, 113, 175, 171, 16, 44, 120, 82, 124, 94, 189, 152, 2, 1, 97, 101, 75, 86, 53, 158, 154, 180, 182, 136, 63, 79, 133, 80, 176, 137, 51]



Dataset B:

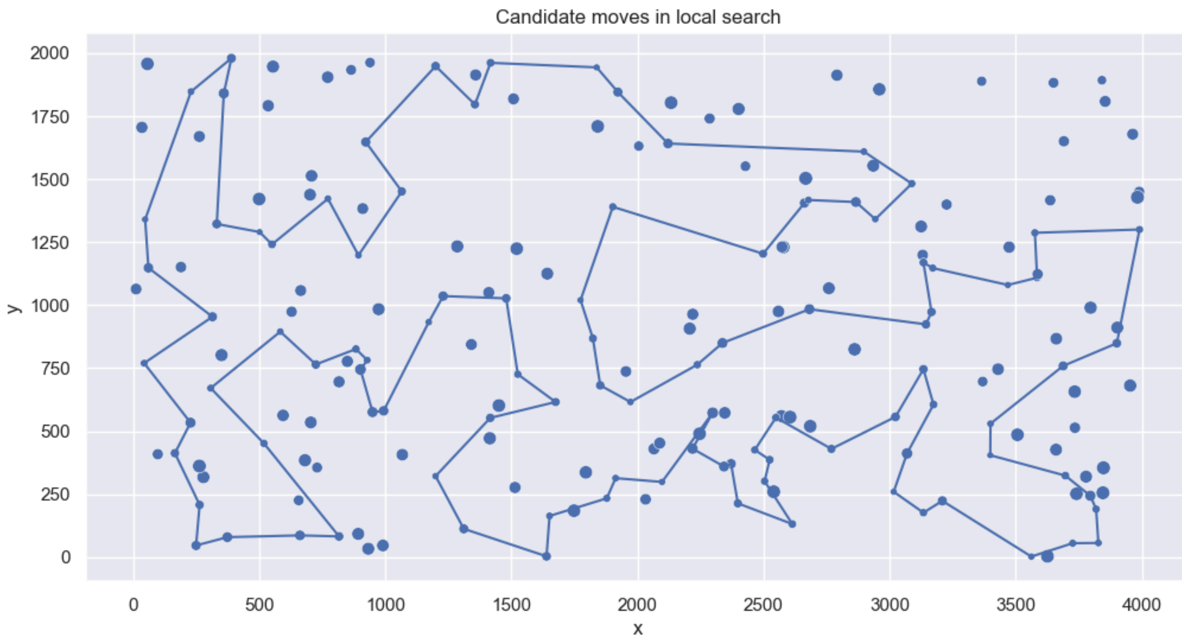
Best solution: [109, 35, 143, 106, 124, 62, 83, 18, 55, 34, 152, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 179, 185, 95, 86, 166, 194, 88, 176, 180, 113, 103, 127, 89, 163, 153, 81, 77, 141, 36, 177, 5, 142, 78, 175, 80, 190, 193, 156, 198, 117, 54, 31, 73, 19, 112, 121, 131, 1, 27, 38, 135, 63, 40, 107, 10, 133, 122, 90, 191, 51, 147, 6, 188, 169, 132, 70, 3, 155, 189, 15, 145, 13, 195, 168, 43, 139, 182, 25, 138, 11, 33, 160, 104, 8, 82, 111, 29, 0]



With k = 50

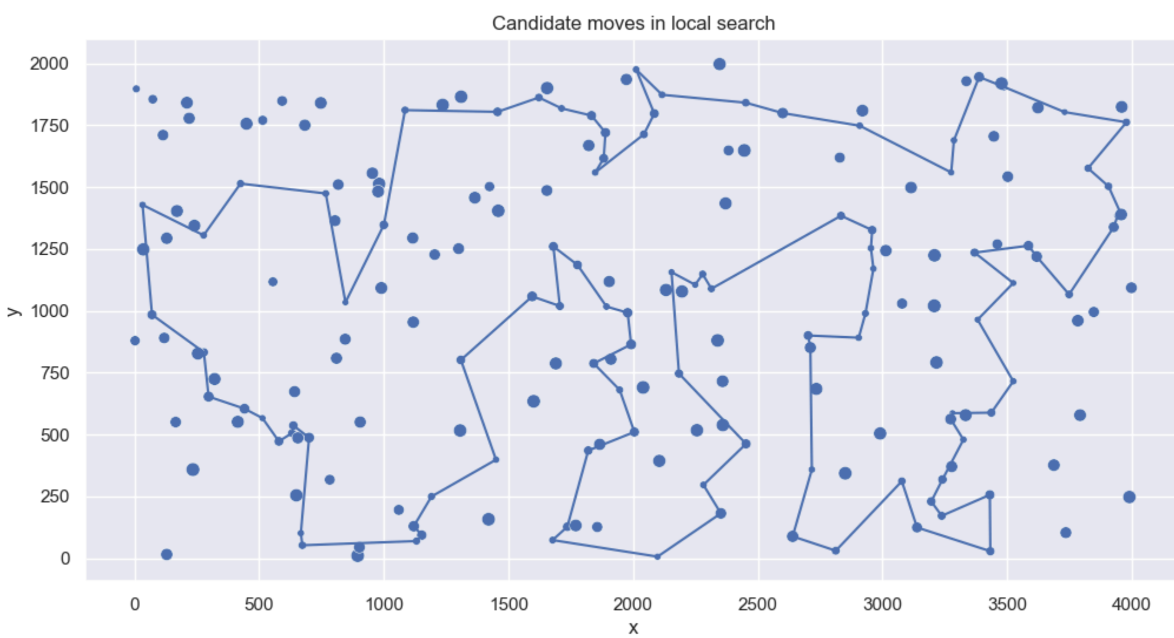
Dataset A:

Best solution: [145, 196, 81, 90, 165, 40, 185, 106, 178, 52, 55, 167, 124, 94, 63, 79, 80, 176, 137, 148, 9, 62, 102, 49, 14, 144, 186, 89, 183, 143, 0, 117, 68, 46, 115, 139, 41, 193, 159, 69, 108, 18, 22, 146, 181, 34, 48, 54, 177, 10, 190, 4, 112, 184, 160, 42, 43, 116, 65, 47, 131, 149, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 189, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 129, 57, 92, 82, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78]



Dataset B:

Best solution: [78, 142, 5, 177, 25, 182, 138, 139, 11, 33, 160, 144, 104, 8, 82, 21, 61, 36, 141, 97, 77, 81, 111, 29, 0, 109, 35, 34, 55, 18, 62, 124, 106, 143, 159, 153, 146, 187, 163, 165, 137, 114, 127, 89, 103, 113, 180, 176, 194, 166, 86, 185, 95, 99, 22, 179, 66, 94, 154, 47, 148, 60, 20, 28, 149, 140, 183, 152, 184, 155, 3, 70, 15, 145, 168, 195, 13, 132, 169, 188, 6, 147, 51, 121, 90, 122, 135, 63, 38, 1, 156, 198, 117, 193, 31, 54, 73, 190, 80, 175]



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	88235.64(81477-96936)	53248.47(48637-59202)
Random, edge, steepest with 20 candidates	81675.91(76766-87313)	50096.05(46561-54425)
Random, edge, steepest with 50 candidates	75681.715(71584-79549)	48664.605(45575-53187)

Average running time

Method	Dataset A	Dataset B
Random generation, edge, steepest	3.3 s	3.17 s
Random, edge, steepest with 10 candidates	4.6 s	5.12 s
Random, edge, steepest with 20 candidates	3.6 s	3.9 s
Random, edge, steepest with 50 candidates	3.3 s	3.6 s

Conclusion:

Looking at both function performance and running times, it is clear that restricting the candidate pool to 10 options is detrimental to both solution quality and speed. The algorithm performs better with more candidates, with the unrestricted baseline version being the most effective approach.

The reason for that is that the algorithm terminates only when there is no improvement. It is harder to find improvement when there are less options to choose from. Hence, the time it takes to finish the algorithm and the quality of the solution is getting worse.

Our thoughts are that to find the solution faster, there should be some early termination condition