Assignment 5: The use of move evaluations (deltas) from previous iterations 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 move evaluations

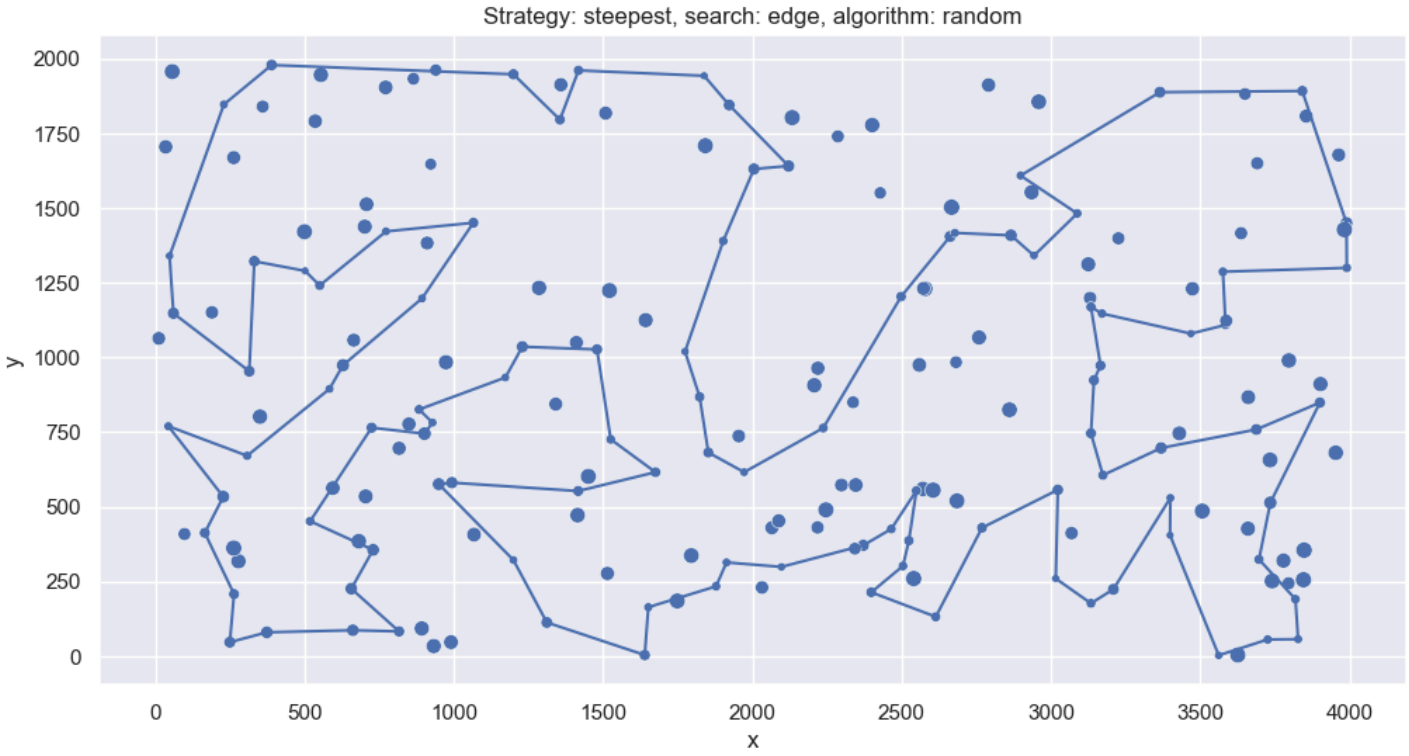
Pseudocode:

```
Initialize statistics tracking for moves
Initialize number of nodes from distance matrix
Create copy of initial solution
Identify selected and non-selected nodes
Calculate initial solution cost

While improvement is possible:
    Generate intra-route neighbors (edge-based)
    Generate inter-route neighbors
    Combine all possible neighbors

    If no neighbors exist:
        Exit loop

    Select best neighbor (minimum cost improvement)

    If no improving neighbor:
        Exit loop

    Update move statistics
```

```
    Update solution:
        - Modify solution based on best neighbor
        - Update selected and non-selected nodes

Return final solution
```

Dataset A:

Best solution: [78, 145, 25, 44, 120, 129, 2, 75, 86, 101, 1, 152, 97, 26, 53, 180, 154, 135, 70, 127, 123, 131, 149, 162, 133, 151, 51, 118, 59, 116, 65, 47, 43, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 34, 160, 42, 5, 115, 46, 139, 41, 193, 159, 181, 146, 22, 18, 108, 117, 0, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 148, 9, 62, 102, 49, 14, 144, 21, 164, 27, 90, 165, 40, 185, 106, 178, 52, 55, 57, 92, 179, 196, 81, 157, 31, 113, 175, 171, 16]



Dataset B:

Best solution: [20, 28, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 143, 159, 41, 111, 82, 87, 21, 8, 104, 144, 0, 35, 109, 29, 160, 33, 138, 182, 11, 139, 168, 195, 145, 15, 189, 155, 3, 70, 161, 13, 132, 169, 188, 6, 134, 147, 51, 121, 90, 122, 133, 10, 107, 40, 63, 135, 38, 1, 117, 193, 31, 54, 73, 190, 80, 45, 142, 175, 78, 5, 177, 36, 61, 79, 91, 141, 77, 81, 153, 187, 163, 89, 127, 103, 113, 180, 176, 194, 166, 86, 95, 185, 179, 94, 47, 148, 60, 23]

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 move evaluations	73892.495(71082-78058)	48257.715(45999-52107)

Average running time

Method	Dataset A	Dataset B
Random generation, edge, steepest	3.3 s	3.17 s
Random, edge, steepest with move evaluations	2.4 s	2.27 s

Conclusion:

With use of move evaluations from previous iterations we can see that we get a good running time decrease, while also getting results very close to the usual local search.