Authors: Weronika Zawadzka 151943 Eliza Czaplicka 151963

Date: November 17, 2024

Best solutions have been checked with the solution checker

Summary Report for w5_move_evals

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.

The goal of the task is to improve the time efficiency of the steepest local search with the use **move evaluations (deltas) from previous iterations** (list of improving moves) using the neighborhood, which turned out to be the best in assignment 3.

For our case, the best neighborhood was 2 edge exchange so we will use it here.

Pseudocode of implemented algorithms

```
Initiate LM – a set of moves that bring improvement ordered from the best to the worst, initially empty
Generate an initial solution x (200 random, take best)
Initiate affectedNodes Set<Integer> as initial solution

MAIN FUNCTION
repeat until no move has been found after checking the whole list LM:
  Evaluate all new moves based on affectedNodes and add improving moves to LM:
    - Consider both intra (2 edges) and inter (2 nodes) exchanges
    - Improving based on delta calculation
    - if delta < 0 then add to LM
    - Considered are also moves with inverted edges

  Browse moves from LM
  newAffectedNodes = {};
  for moves m from LM starting from the best:
    check if move is valid/invalid or should be skipped

    if valid then:
      apply,
      remove from LM,
      add affected nodes to newAffectedNodes
```

break;
if invalid then remove from LM;
if should be skipped then skip;

affectedNodes = newAffectedNodes;

CHECKING VALIDITY

1. Check if required nodes are still in the solution:

   For inter-route moves:
   prevNode, node1, and nextNode must be in the solution.
   node2 must not be in the solution.
   For intra-route moves:
   prevNode, node1, node2, and nextNode must all be in the solution.
   If any condition fails → INVALID (0).

2. Check if edges of interest between those nodes still in solution -> if not, invalid

   For inter-route moves:
   Check adjacency:
   prevNode ↔ node1
   node1 ↔ nextNode
   For intra-route moves:
   prevNode ↔ node1
   node2 ↔ nextNode
   If adjacency fails → INVALID (0).

3. Check if edges have same order -> if not, skip

   For inter-route moves:
   Ensure prevNode, node1, and nextNode are in the correct relative order.
   If prevNode -> node1 -> nextNode is reversed → SKIP (2).
   For intra-route moves:
   Ensure:
   prevNode -> node1 and node2 ->nextNode are in the correct order.
   Start index (node1) must be less than the end index (node2).
   If the relative order is reversed → SKIP (2).

4. Else: Move is VALID (1).

Summary performance of each method

Summary for Execution Time (ms)

Method	A	B
Steepest2edgesMoveEvals	87.575 (49 - 482)	93.24 (48 - 475)
Steepest2edgesRandom	929.79 (820 - 1061)	926.84 (788 - 1084)

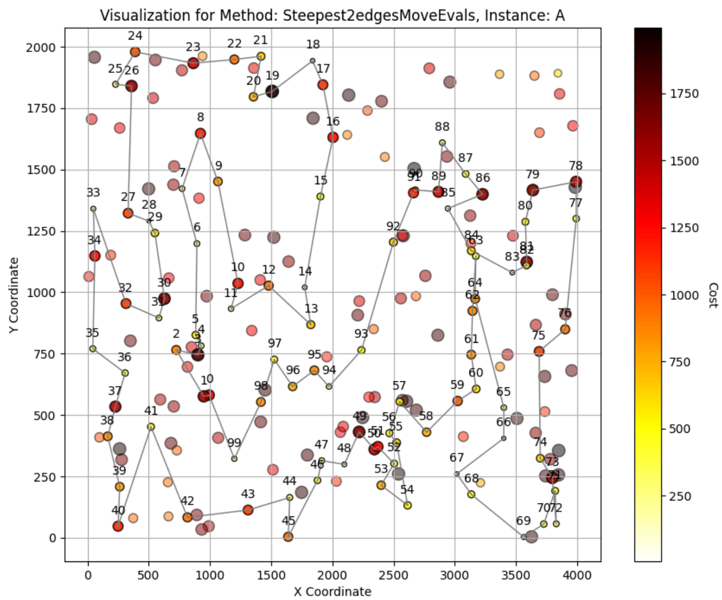
Summary for Objective Function Value

Method	A	B
Steepest2edgesMoveEvals	75485.68 (72800 - 78951)	49734.125 (46217 - 52879)
Steepest2edgesRandom	75337.8 (72622 - 78771)	49424.465 (46854 - 52233)

2D visualisations of best solutions

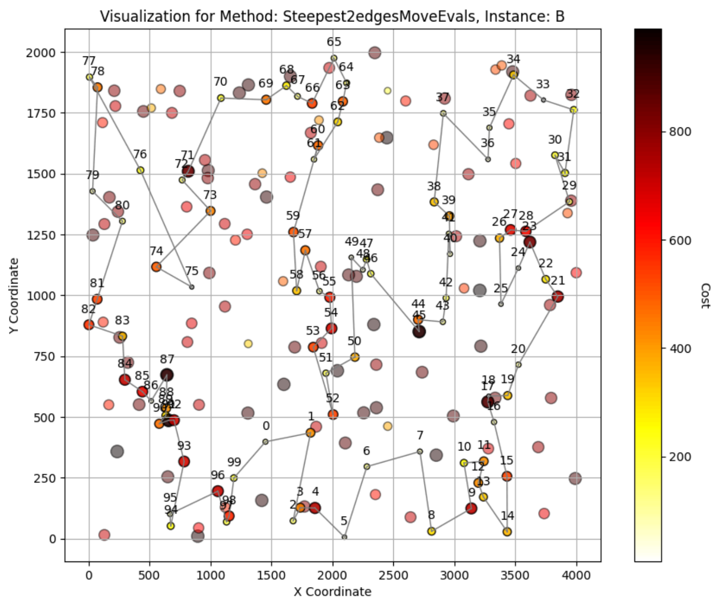
Instance: A

Method: Steepest2edgesMoveEvals with score=72800



Instance: B

Method: Steepest2edgesMoveEvals with score=46217



Best solutions, indices

Instance: A

Method: Steepest2edgesMoveEvals

Lowest Objective Function Value (f_val): 72800

Solution:

149, 131, 43, 47, 65, 116, 115, 139, 68, 46, 118, 59, 51, 80, 176, 137, 23, 89, 183, 153, 0, 143, 117, 140, 108, 18, 69, 159, 193, 41, 5, 42, 181, 22, 146, 34, 160, 48, 54, 177, 10, 184, 112, 127, 135, 70, 154, 180, 53, 121, 100, 26, 101, 86, 75, 1, 97, 152, 2, 129, 92, 57, 55, 106, 52, 145, 78, 120, 44, 16, 171, 113, 175, 56, 31, 196, 81, 90, 27, 39, 165, 119, 40, 185, 178, 49, 138, 14, 144, 102, 62, 9, 148, 94, 63, 79, 133, 151, 162, 123

Instance: B

Method: Steepest2edgesMoveEvals

Lowest Objective Function Value (f_val): 46217

Solution:

177, 21, 36, 61, 91, 141, 77, 153, 187, 165, 163, 103, 89, 127, 137, 114, 113, 180, 176, 194, 166, 172, 179, 22, 185, 86, 95, 130, 99, 94, 148, 47, 60, 20, 28, 140, 183, 152, 34, 55, 62, 18, 124, 106, 143, 159, 35, 109, 0, 29, 111, 8, 82, 104, 144, 160, 33, 11, 138, 139, 195, 168, 145, 15, 3, 70, 132, 169, 188, 6, 147, 191, 90, 51, 131, 121, 122, 40, 107, 63, 135, 38, 27, 1, 156, 198, 117, 151, 54, 31, 193, 164, 73, 136, 80, 190, 45, 175, 78, 5

Conclusions

The final average objective function values are almost the same but the time was dramatically reduced by using previous deltas with move list: **Steepest2edgesMoveEvals** time across both methods is 87.575 ms for Instance A and 93.24 ms for Instance B compared to **Steepest2edgesRandom** with approximately 930 ms for both instances. We aimed for the same results in terms of objective function however our implementation achieved a little worse (by ~150 on instance A and ~300 on instance B) ones. Nonetheless we can say **Steepest2edgesMoveEvals** offered dramatic speed-ups without significant degradation in solution quality.