Report

Team members:

- Sofya Aksenyuk, 150284
- Uladzimir Ivashka, 150281

Problem Description

Given a set of nodes, each characterized by their (x, y) coordinates in a plane and an associated cost, the challenge is to select exactly 50% of these nodes and form a Hamiltonian cycle.

The goal is to minimize the sum of the total length of the path plus the total cost of the selected nodes.

Distances between nodes are computed as Euclidean distances and rounded to the nearest integer.

Methodology

Steepest Local Search with the use of move evaluations (deltas) from previous iterations

Steepest Local Search systematically examines all possible moves within the neighborhood, both intra-route and inter-route, and selects the move that results in the best improvement in the objective function value.

It aims to find the absolute best move at each step.

Steepest Local Search with Move Evaluations begins with an initial solution and iteratively explores neighboring solutions to identify potential moves.

During each iteration, the algorithm explores neighboring solutions and calculates the delta for each move.

By evaluating the moves in this manner, the algorithm can prioritize and select the most promising moves that lead to improvements in solution quality.

The use of move evaluations from previous iterations allows the algorithm to adapt its search strategy based on the historical performance of moves.

Moves that have previously demonstrated a positive impact on solution quality are given higher priority, guiding the search towards more promising regions of the solution space.

Source code

Link: Source Code

Pseudocode

Steepest Local Search with the use of move evaluations (deltas) from previous iterations

```
FUNCTION SteepestLocalSearchDeltas(Solution, DistanceMatrix, Costs)

    LM = SortedSet() // to store moves and their deltas
    Calculate the size of the solution and the total number of nodes
    Improved = True

    WHILE (Improved is True):
        Improved = False

        // Inter moves
        FOR each Move in combination of inner node and outer node:
            Delta = (calculate delta for inter move)
            IF (Delta > 0): //brings improvement
                LM.add((Delta, Move, MoveInfo))

        // Intra moves
        FOR each Move in combination of two edges in solution:
            Delta = (calculate delta for intra edge move)
            IF (Delta > 0): //brings improvement
                LM.add((Delta, Move, MoveInfo))

        // Process LM
        FOR each Move in LM:
            ToApply, ToStore = CheckMoveValidity(Solution, Move)
            IF (ToApply is True):
                Improved = True
                Solution = (apply move)
                LM.remove(Move) // move is made, remove it
            ELSE IF (ToStore is not True):
                LM.remove(Move) // move is no valid
            IF Improved:
                Break loop // if move already found and made

    RETURN solution
```

```
FUNCTION CheckMoveValidity(Solution, Move)
    Decompose Move into MoveType, MoveNodes, and AdjacentNodes

    IF (MoveType == 'inter'):
        ExternalNode, InternalNode = MoveNodes
        AdjacentNodePrev, AdjacentNodeNext = AdjacentNodes

        IF (InternalNode in Solution and ExternalNode not in Solution):
            // Check existence and order of edges involving InternalNode
            EdgePrevExists = ((AdjacentNodePrev, InternalNode) forms an edge in
correct order in Solution)
            EdgeNextExists = ((InternalNode, AdjacentNodeNext) forms an edge in
correct order in Solution)
            RETURN (EdgePrevExists and EdgeNextExists, not (EdgePrevExists and
EdgeNextExists))
        ELSE:
            RETURN (False, False)

    IF (MoveType == 'intra'):
        Node1, Node2 = MoveNodes
        AdjacentNode1, AdjacentNode2 = AdjacentNodes

        // Check existence and order of edges for intra move
        Edge1Exists = ((Node1, AdjacentNode1) forms an edge in correct order in
Solution)
        Edge2Exists = ((Node2, AdjacentNode2) forms an edge in correct order in
Solution)
        RETURN (Edge1Exists and Edge2Exists, not (Edge1Exists and Edge2Exists))
```

Computational Experiments

Results

Table of Cost

Algorithm	TSPA	TSPB	TSPC	TSPD
0 RandomSearch	264715.690 (234787.0 – 289096.0)	265095.315 (238995.0 – 287788.0)	214163.21 (192811.0 – 232188.0)	217922.530 (195986.0 – 246928.0)
1 NearestNeighbor	86319.43 (83561.0 - 93749.0)	77688.285 (75928.0 – 79791.0)	56709.06 (53466.0 – 60369.0)	52472.030 (48240.0 – 57939.0)
2 GreedyCycle	78049.310 (75990.0 – 81934.0)	71717.795 (68973.0 – 78237.0)	56404.07 (53723.0 – 58868.0)	55334.72 (50717.0 – 61896.0)
3 Greedy2Regret	117178.570 (110218.0 -124855.0)	121592.715 (111115.0 – 131138.0)	69547.98 (66114.0 – 73603.0)	71900.575 (66024.0 – 76887.0)
4 Greedy2RegretWeighted	75953.435 (74701.0 – 78510.0)	71428.365 (69147.0 – 77017.0)	54217.24 (51747.0 – 58087.0)	52285.360 (47629.0 – 57787.0)
5 Greedy-edges-Random	77965.71 (75112.0 - 82142.0)	71018.74 (68315.0 - 74769.0)	51652.62 (49206.0 - 54174.0)	48705.455 (45913.0 - 51873.0)
6 Greedy-edges-GreedyHeuristic	75438.075 (74521.0 - 77298.0)	70388.815 (67808.0 - 76131.0)	53714.495 (51034.0 - 56971.0)	51525.635 (47281.0 - 57524.0)
7 Greedy-nodes-Random	90554.405 (83565.0 - 100644.0)	85537.285 (77218.0 - 93542.0)	63718.49 (56594.0 - 71397.0)	61963.345 (54964.0 - 69495.0)
8 Greedy-nodes-GreedyHeuristic	75512.725 (74521.0 - 77471.0)	70688.61 (68572.0 - 76529.0)	53967.21 (51195.0 - 56739.0)	51145.425 (47368.0 - 57605.0)
9 Steepest-edges-GreedyHeuristic	75391.6 (74541.0 - 77063.0)	70215.965 (68122.0 - 75907.0)	53529.02 (50972.0 - 57024.0)	51093.845 (46915.0 - 57412.0)
10 Steepest-nodes-Random	93224.675 (84578.0 - 101323.0)	88096.545 (80827.0 - 96390.0)	65743.78 (58924.0 - 77555.0)	64443.6 (55944.0 - 78053.0)
11 Steepest-nodes-GreedyHeuristic	75606.625 (74630.0 - 76953.0)	70758.4 (68572.0 - 76545.0)	53807.135 (51203.0 - 57026.0)	51203.305 (46990.0 - 55945.0)
12 Steepest-edges-Random	78307.45 (75390.0 - 83932.0)	71637.29 (68186.0 - 77703.0)	51692.4 (49183.0 - 54617.0)	48647.455 (46150.0 - 51579.0)
13 Steepest-CandidateMovesEdges-Random	80856.86 (76709.0 - 85935.0)	73804.335 (69986.0 - 80091.0)	51680.02 (49141.0 - 54538.0)	48294.905 (45672.0 - 52381.0)
14 Steepest-PreviousDeltas-Random	79913.365 (76798.0 - 87582.0)	73141.7 (68884.0 - 78988.0)	53064.8 (49346.0 - 57601.0)	49794.255 (46581.0 - 55387.0)

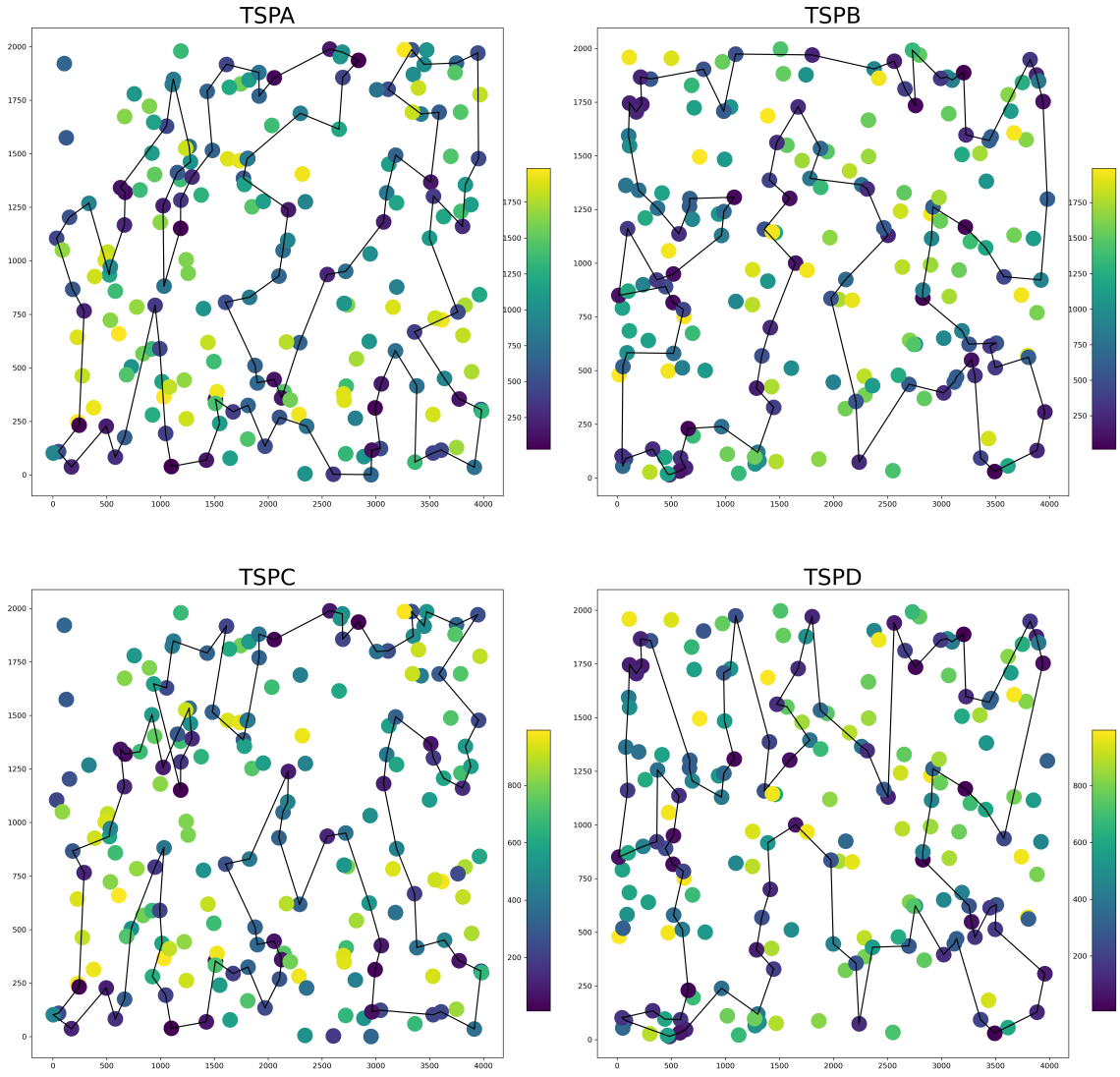
Table of Time

Algorithm	TSPA	TSPB	TSPC	TSPD
0 Greedy-edges-Random	7.682 (6.255 - 9.005)	7.989 (6.794 - 9.754)	7.739 (6.556 - 9.202)	6.699 (4.754 - 12.083)
1 Greedy-edges-GreedyHeuristic	0.269 (0.073 - 0.655)	0.418 (0.202 - 0.92)	0.315 (0.069 - 0.831)	0.366 (0.102 - 0.995)
2 Greedy-nodes-Random	4.005 (3.243 - 5.092)	4.126 (3.348 - 5.451)	3.946 (3.129 - 5.034)	4.469 (3.086 - 9.898)
3 Greedy-nodes-GreedyHeuristic	0.243 (0.082 - 0.802)	0.291 (0.145 - 0.581)	0.242 (0.073 - 0.663)	0.284 (0.083 - 0.765)
4 Steepest-edges-GreedyHeuristic	0.538 (0.073 - 1.405)	0.883 (0.337 - 1.591)	0.648 (0.065 - 1.511)	0.73 (0.133 - 2.109)
5 Steepest-nodes-Random	14.777 (11.815 - 18.468)	14.972 (12.168 - 18.308)	14.376 (11.243 - 18.885)	14.716 (11.481 - 19.12)
6 Steepest-nodes-GreedyHeuristic	0.335 (0.074 - 1.124)	0.669 (0.345 - 1.219)	0.578 (0.077 - 1.63)	0.628 (0.145 - 1.347)
7 Steepest-edges-Random	12.176 (10.589 - 14.762)	12.516 (11.009 - 14.349)	11.872 (10.033 - 13.814)	12.583 (10.98 - 14.19)
8 Steepest-CandidateMovesEdges-Random	1.448 (1.205 - 2.451)	1.476 (1.217 - 1.761)	1.437 (1.203 - 1.788)	1.364 (1.078 - 2.015)
9 Steepest-PreviousDeltas-Random	6.517 (5.292 - 7.677)	6.523 (5.425 - 8.05)	6.448 (5.265 - 7.948)	6.485 (5.131 - 7.811)

Best Solutions Plots

See plots: Plots

Steepest-PreviousDeltas-Random



Best solution among all methods so far

TSPA

[42, 89, 94, 12, 72, 190, 98, 66, 156, 6, 24, 141, 144, 87, 79, 194, 21, 171, 154, 81, 62, 108, 15, 117, 53, 22, 195, 55, 36, 132, 128, 25, 181, 113, 74, 163, 61, 71, 20, 64, 185, 96, 27, 116, 147, 59, 143, 159, 164, 178, 19, 0, 149, 50, 121, 91, 114, 4, 77, 43, 192, 175, 153, 88, 127, 186, 45, 167, 101, 60, 126, 174, 199, 41, 177, 1, 75, 189, 109, 130, 152, 11, 48, 106, 26, 119, 134, 99, 135, 51, 5, 112, 73, 31, 95, 169, 8, 80, 14, 111]

Cost: 74541.0

TSPB

[158, 162, 150, 44, 117, 196, 192, 21, 142, 130, 174, 51, 91, 70, 140, 148, 141, 53, 69, 115, 82, 63, 8, 14, 16, 172, 95, 163, 182, 2, 5, 34, 183, 197, 179, 31, 101, 42, 38, 103, 131, 121, 24, 127, 143, 122, 92, 26, 66, 169, 99, 50, 154, 134, 25, 36, 165, 37, 137, 88, 55, 4, 153, 145, 157, 80, 57, 0, 135, 198, 190, 19, 29, 33, 136, 61, 73, 185, 132, 18, 52, 12, 107, 139, 193, 119, 59, 71, 166, 85, 64, 147, 159, 89, 129, 58, 171, 72, 114, 67]

Cost: 68122.0

TSPC

[81, 171, 108, 62, 15, 117, 53, 22, 55, 195, 74, 163, 113, 132, 128, 40, 164, 178, 19, 35, 69, 0, 149, 50, 121, 91, 114, 175, 2, 4, 77, 43, 192, 150, 199, 39, 174, 137, 41, 177, 1, 75, 189, 109, 130, 152, 11, 160, 106, 48, 92, 26, 119, 134, 139, 95, 169, 110, 8, 80, 31, 73, 89, 42, 94, 12, 72, 98, 156, 172, 6, 66, 190, 112, 5, 51, 135, 99, 101, 9, 60, 167, 153, 88, 127, 45, 186, 170, 129, 157, 21, 194, 79, 87, 141, 144, 102, 44, 133, 154]

Cost: 50972.0

TSPD

[87, 147, 159, 64, 129, 89, 58, 171, 72, 114, 85, 166, 28, 59, 119, 193, 71, 44, 162, 150, 117, 196, 192, 21, 138, 142, 130, 161, 174, 188, 140, 148, 141, 53, 96, 32, 113, 69, 115, 82, 63, 8, 14, 84, 139, 97, 107, 12, 52, 132, 18, 16, 172, 95, 19, 190, 198, 135, 128, 66, 169, 0, 57, 99, 92, 122, 143, 179, 121, 127, 24, 50, 112, 154, 134, 25, 36, 194, 123, 165, 37, 146, 137, 88, 55, 4, 153, 80, 157, 145, 79, 136, 61, 73, 185, 47, 189, 170, 181, 187]

Cost: 46915.0

Conclusions

Cost Comparison

Among all TSP instances Steepest-PreviousDeltas-Random method shows **slightly** higher costs than Steepest-edges-Random

Time Efficiency

A notable reduction in computation time is observed with Steepest-PreviousDeltas-Random across all TSP instances when compared to Steepest-edges-Random

This reduction aligns with the goal of improving time efficiency through the use of move evaluations from previous iterations

However, it still takes much longer in comparison with Steepest-CandidateMovesEdges-Random