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

Multiple Start Local Search and Iterated Local Search

MSLS involves initiating the local search algorithm from multiple initial solutions, exploring different regions of the solution space simultaneously.

This approach aims to diversify the search process, increasing the chances of finding high-quality solutions.

On the other hand, ILS is an iterative refinement technique that involves repeatedly applying a local search algorithm to the best solution found so far.

After reaching a local optimum, ILS introduces perturbations to escape the current solution and explores new regions of the solution space.

This process helps to balance exploration and exploitation, allowing the algorithm to escape suboptimal solutions and potentially discover better solutions.

Both MSLS and ILS enhance the cost- and time-efficiency of local search algorithms.

Source code

Link: Source Code

Pseudocode

Multiple Start Local Search

```
FUNCTION MSLS(DistanceMatrix, Costs, NumRuns=200)

    BestTotalCost = Inf
    BestSolution = None

    FOR each Run in Range(NumRuns):
        Solution = RandomSearch()
        Solution = SteepestLocalSearch(Solution, DistanceMatrix, Costs)
        TotalCost = GetTotalCost(Solution, DistanceMatrix, Costs)
        IF (TotalCost < BestTotalCost):
            BestTotalCost = TotalCost
            BestSolution = Solution

    RETURN BestSolution, BestTotalCost
```

Iterated Local Search

```
FUNCTION ILS(DistanceMatrix, Costs, EndTime):
    StartTime = time()
    BestSolution = RandomSearch()
    BestSolution = SteepestLocalSearch(BestSolution, DistanceMatrix, Costs)
    BestTotalCost = GetTotalCost(BestSolution, DistanceMatrix, Costs)
    Counter = 0

    WHILE (time() - StartTime < EndTime):
        SolutionPerturbed = Perturb(BestSolution)
        NewSolution = SteepestLocalSearch(SolutionPerturbed, DistanceMatrix,
Costs)
        Counter += 1
        NewTotalCost = GetTotalCost(NewSolution, DistanceMatrix, Costs)

        IF (NewTotalCost < BestTotalCost):
            BestTotalCost = NewTotalCost
            BestSolution = NewSolution

    RETURN BestSolution, BestTotalCost, Counter

FUNCTION Perturb(Solution):

    Randomly choose and apply to solution one of the actions:

        1) Randomly select n cities from the current solution and insert them into
random positions within the solution
        2) Divide the solution into four segments of random size, and then
rearrange these segments in a specific order
        3) Select a sub-path of random length (between 5% to 15% of the total path
length) and shuffle the order of cities within this sub-path
        4) Select a sub-path of random length, remove this sub-path from its
original position and insert it at a different random location in the solution
        5) Remove k edges in the solution, reconnect the resulting segments in a
different order
```

Computational Experiments

Results

Table of Cost

Algorithm	TSPA	TSPB	TSPC	TSPD
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
MSLS	75178.125 (74562 - 75588)	68299.225 (67595 - 68770)	49113.225 (48452 - 49647)	45531.725 (44622 - 46017)
ILS	73078.1 (72855 - 73279)	66442.5 (66117 - 66770)	47136.4 (46811 - 47604)	43407.25 (43207 - 43949)

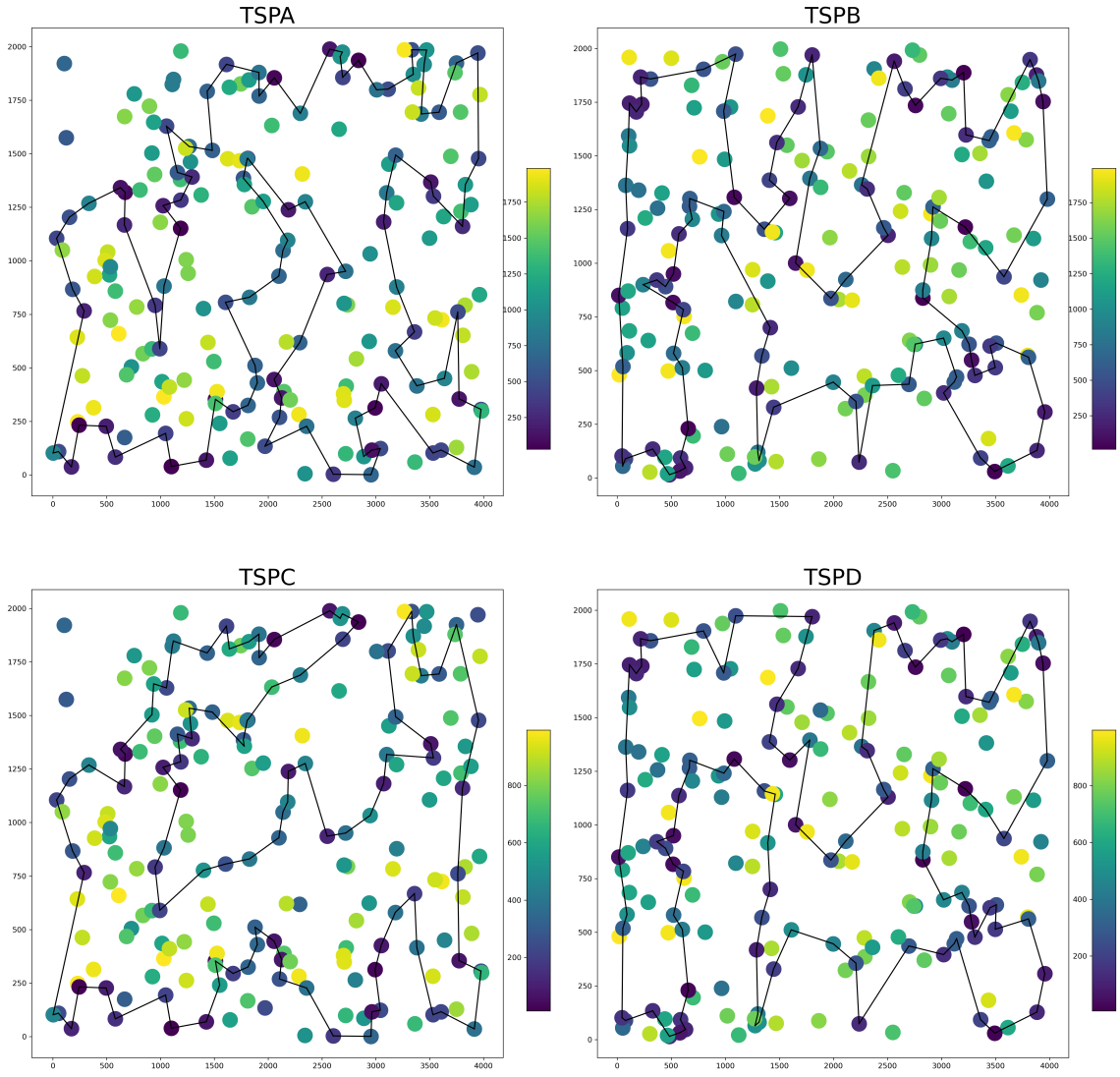
Table of Time

Algorithm	TSPA	TSPB	TSPC	TSPD
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
MSLS	1263.926 (1233.821 - 1307.715)	1310.493 (1272.85 - 1428.453)	1266.808 (1237.736 - 1355.937)	1268.967 (1237.212 - 1352.452)
ILS	1264.199 (1264.006 - 1264.605)	1310.183 (1310.003 - 1310.458)	1267.235 (1267.003 - 1267.694)	1269.149 (1269.014 - 1269.642)

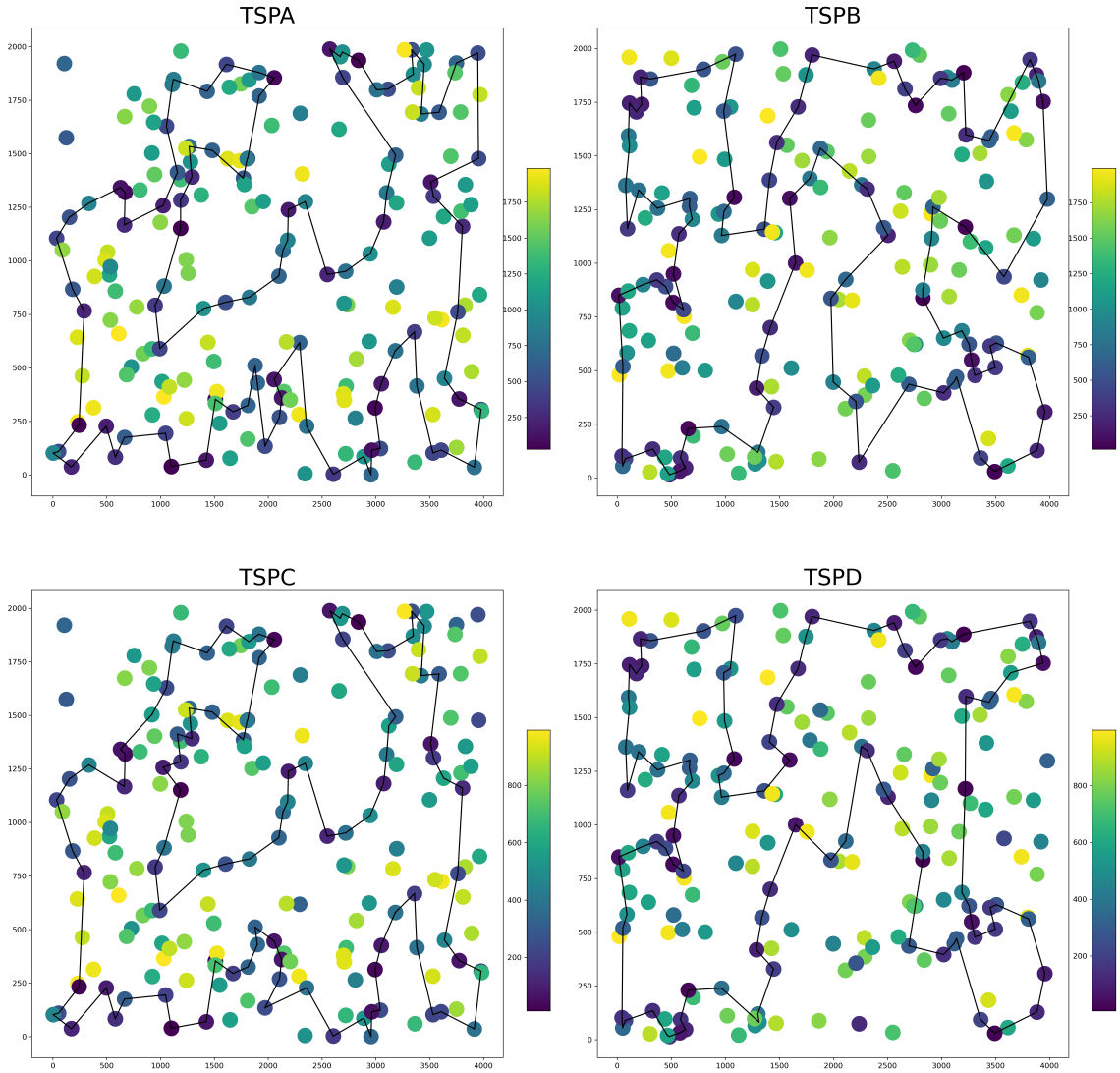
Best Solutions Plots

See plots: Plots

MSLS



ILS



Number of runs of Steepest Local Search in ILS

instance	TSPA	TSPB	TSPC	TSPD
No. iterations ILS	4197.75 (3991 - 4326)	4160.4 (4054 - 4370)	4206.3 (3835 - 4365)	4184.95 (4009 - 4348)

Best solution among all methods so far

TSPA

[48, 106, 160, 11, 152, 130, 119, 109, 189, 75, 1, 177, 41, 137, 199, 192, 175, 114, 4, 77, 43, 121, 91, 50, 149, 0, 19, 178, 164, 159, 143, 59, 147, 116, 27, 96, 185, 64, 20, 71, 61, 163, 74, 113, 195, 53, 62, 32, 180, 81, 154, 144, 141, 87, 79, 194, 21, 171, 108, 15, 117, 22, 55, 36, 132, 128, 145, 76, 161, 153, 88, 127, 186, 45, 167, 101, 99, 135, 51, 112, 66, 6, 172, 156, 98, 190, 72, 12, 94, 89, 73, 31, 111, 14, 80, 95, 169, 8, 26, 92]

Cost: 72855.0

TSPB

[166, 59, 119, 193, 71, 44, 196, 117, 150, 162, 158, 67, 156, 91, 70, 51, 174, 140, 148, 141, 130, 142, 53, 69, 115, 82, 63, 8, 16, 18, 29, 33, 19, 190, 198, 135, 95, 172, 163, 182, 2, 5, 34, 183, 197, 31, 101, 38, 103, 131, 24, 127, 121, 179, 143, 122, 92, 26, 66, 169, 0, 57, 99, 50, 112, 154, 134, 25, 36, 165, 37, 137, 88, 55, 153, 80, 157, 145, 79, 136, 73, 185, 132, 52, 139, 107, 12, 189, 170, 181, 147, 159, 64, 129, 89, 58, 171, 72, 114, 85]

Cost: 66117.0

TSPC

[61, 113, 74, 163, 155, 62, 32, 180, 81, 154, 102, 144, 141, 87, 79, 194, 21, 171, 108, 15, 117, 53, 22, 195, 55, 36, 132, 128, 145, 76, 161, 153, 88, 127, 186, 45, 167, 101, 99, 135, 51, 5, 112, 66, 6, 172, 156, 98, 190, 72, 12, 94, 89, 73, 31, 95, 169, 110, 8, 26, 92, 48, 106, 160, 11, 152, 130, 119, 109, 189, 75, 1, 177, 41, 137, 199, 192, 43, 77, 4, 114, 91, 121, 50, 149, 0, 19, 178, 164, 159, 143, 59, 147, 116, 27, 96, 185, 64, 20, 71]

Cost: 46811.0

TSPD

[79, 145, 157, 80, 153, 4, 55, 88, 36, 25, 134, 154, 123, 165, 37, 137, 99, 92, 122, 143, 179, 121, 127, 24, 131, 103, 38, 101, 31, 197, 183, 34, 5, 128, 66, 169, 135, 198, 190, 19, 95, 172, 16, 8, 63, 82, 115, 69, 113, 53, 142, 130, 141, 148, 140, 188, 174, 51, 70, 91, 156, 3, 67, 158, 162, 150, 117, 196, 44, 71, 193, 119, 59, 166, 85, 114, 72, 171, 58, 89, 129, 64, 159, 147, 181, 170, 47, 189, 109, 12, 107, 97, 139, 52, 18, 132, 185, 73, 61, 136]

Cost: 43207.0

Conclusions

Cost Efficiency

Both MSLS and ILS consistently outperform the Steepest Local Search variants in terms of cost efficiency across all instances.

ILS on average tends to provide slightly lower costs compared to MSLS.

Time Efficiency

Both MSLS and ILS exhibit higher computational times compared to Steepest Local Search variants.

This is expected as they involve multiple iterations and exploration steps.