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.

Methodologies

Greedy Local Search

This algorithm repeatedly makes random selections of moves (intra-route or inter-route) to explore and improve a solution.

It aims to diversify the search and potentially find better solutions by introducing randomness into the order of exploration.

Steepest Local Search

This algorithm 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.

Source code

Link: Source Code

Pseudocode

Greedy Local Search

```
Function GreedyLocalSearch(DistanceMatrix, Costs):
    CurrentSolution = (generate initial solution using
RandomSearch/GreedyHeuristic)
    Improved = True

    While Improved:
        NeighborsNodeSwap = (generate all possible inter/intra node swaps)
        NeighborsEdgeSwap = (generate all possible inter/intra node swaps)
        Neighbors = NeighborsNodeSwap + NeighborsEdgeSwap
        Neighbors = Shuffle(Neighbors)
        Improved = False

        For each Neighbor in Neighbors:
            If (Neighbor is node swap):
                Delta = CalculateNodeSwapDelta(CurrentSolution, DistanceMatrix,
Costs)
            If (Neighbor is edge swap):
                Delta = CalculateEdgeSwapDelta(CurrentSolution, DistanceMatrix,
Costs)
            If Delta > 0:
                CurrentSolution = Neighbor
                Improved = True
                Break

    Return CurrentSolution
```

Steepest Local Search

```
Function SteepestLocalSearch(DistanceMatrix, Costs):
    CurrentSolution = (generate initial solution using
RandomSearch/GreedyHeuristic)
    Improved = True

    While Improved:
        NeighborsNodeSwap = (generate all possible inter/intra node swaps)
        NeighborsEdgeSwap = (generate all possible inter/intra node swaps)
        Neighbors = NeighborsNodeSwap + NeighborsEdgeSwap
        Improved = False

        BestNeighbor = CurrentSolution
        BestDelta = 0

        For each Neighbor in Neighbors:
            If (Neighbor is node swap):
                Delta = CalculateNodeSwapDelta(CurrentSolution, DistanceMatrix,
Costs)
            If (Neighbor is edge swap):
                Delta = CalculateEdgeSwapDelta(CurrentSolution, DistanceMatrix,
Costs)
            If Delta > BestDelta:
                BestNeighbor = Neighbor
                BestDelta = Delta

        If BestNeighbor != CurrentSolution:
            CurrentSolution = BestNeighbor
            Improved = True

    Return CurrentSolution
```

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-intra-GreedyHeuristic	75675.58 (74610.0 - 77834.0)	70978.11 (68810.0 - 76539.0)	54261.745 (51470.0 - 57922.0)	51988.515 (47663.0 - 57520.0)
6	Greedy-intra-Random	125580.04 (113422.0 - 135853.0)	121994.41 (109150.0 - 136903.0)	75300.81 (68287.0 - 82025.0)	73293.085 (67060.0 - 79235.0)
7	Greedy-inter-GreedyHeuristic	75425.235 (74521.0 - 76788.0)	70331.65 (68006.0 - 75287.0)	53682.4 (50744.0 - 57711.0)	51201.255 (47012.0 - 57524.0)
8	Greedy-inter-Random	77913.7 (74909.0 - 82674.0)	70990.945 (68461.0 - 75250.0)	51693.315 (48964.0 - 54742.0)	48766.42 (45706.0 - 51972.0)
9	Steepest-intra-GreedyHeuristic	75673.415 (74564.0 - 77834.0)	70801.835 (68633.0 - 76107.0)	53933.495 (51696.0 - 57918.0)	51690.78 (47401.0 - 56629.0)
10	Steepest-intra-Random	123909.335 (113745.0 - 133165.0)	120369.82 (108269.0 - 129887.0)	74236.495 (69013.0 - 80418.0)	72433.845 (66007.0 - 80361.0)
11	Steepest-inter-GreedyHeuristic	75448.855 (74534.0 - 77063.0)	70343.165 (67991.0 - 75907.0)	53766.17 (50972.0 - 57024.0)	51417.055 (47281.0 - 56142.0)
12	Steepest-inter-Random	78166.91 (75676.0 - 82885.0)	71787.695 (68111.0 - 77246.0)	51859.81 (49285.0 – 55835.0)	48470.665 (45658.0 - 52566.0)

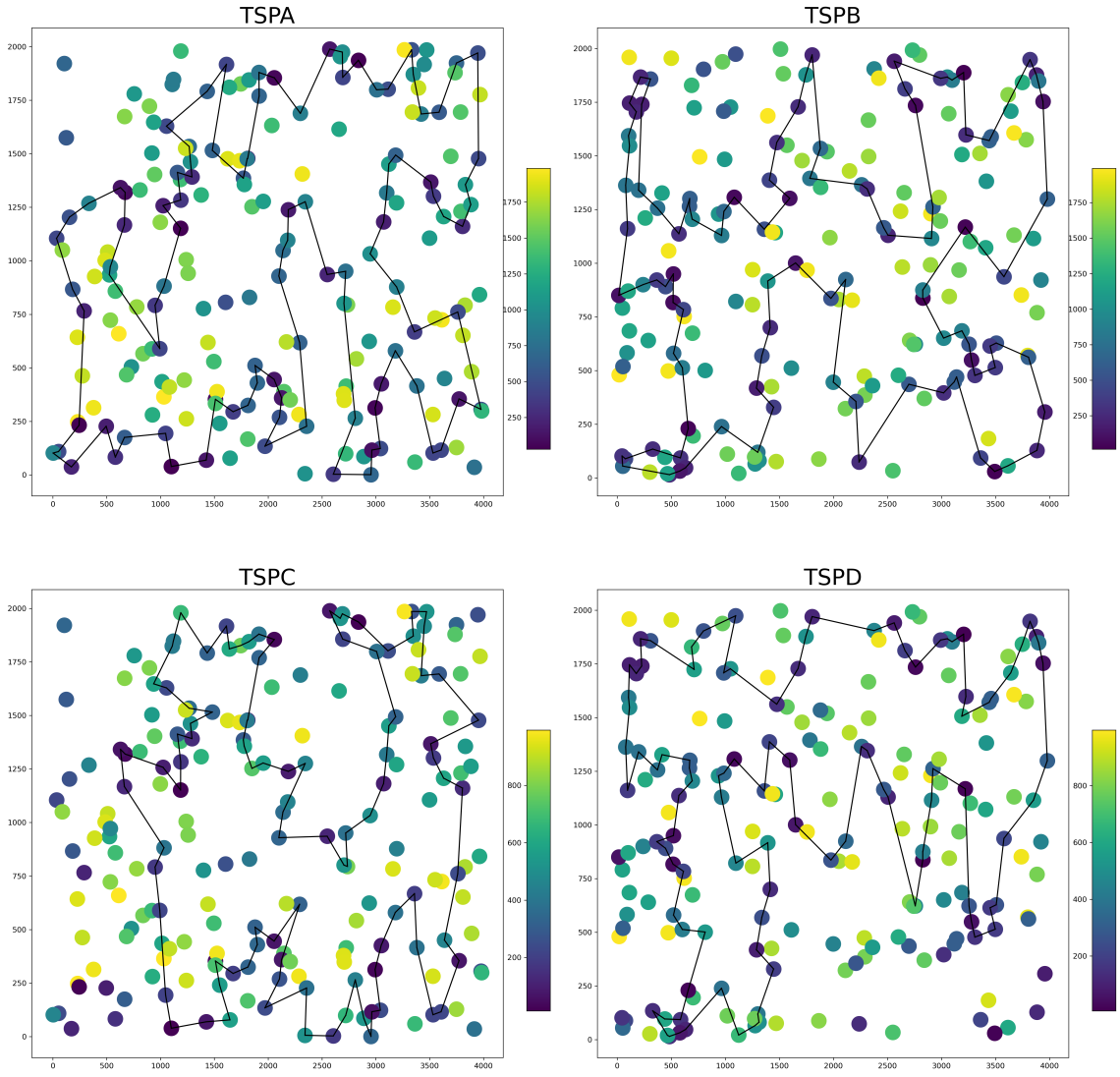
Table of Time

	Algorithm	TSPA	TSPB	TSPC	TSPD
0	Greedy-intra-GreedyHeuristic	0.9232(0.3357 – 1.1102)	0.7168 (0.4455 – 0.9193)	0.796 (0.352 – 1.1734)	0.974 (0.436- 1.2247)
1	Greedy-intra-Random	2.1334 (1.5299 – 2.3991)	1.9636 (1.0065 – 2.0241)	1.8150 (1.0238 – 2.1812)	2.2499(1.6373- 2.5309)
2	Greedy-inter-GreedyHeuristic	0.7537 (0.435 – 1.0116)	0.8029 (0.649 – 1.2184)	0.9646(0.7446 – 1.3169)	0.1693 (0.0604 – 0.3824)
3	Greedy-inter-Random	4.9058 (4.12 - 5.808)	5.1961 (4.416 - 6.222)	4.9152 (4.144 - 5.932)	5.0008 (4.07 - 6.049)
4	Steepest-intra-GreedyHeuristic	0.5088 (0.097 - 0.975)	0.7702 (0.29 - 1.346)	0.4663 (0.098 - 1.276)	0.6465 (0.192 - 1.369)
5	Steepest-intra-Random	13.3413 (11.47 - 14.974)	13.4566 (11.687 - 15.82)	13.4207 (11.314 - 15.922)	13.4603 (11.31 - 16.297)
6	Steepest-inter-GreedyHeuristic	0.9068 (0.121 - 2.233)	1.4897 (0.589 - 2.549)	1.0014 (0.24 - 2.461)	1.2668 (0.241 - 2.365)
7	Steepest-inter-Random	23.7035 (20.867 - 26.497)	24.1494 (21.11 - 28.023)	22.11 (20.15 – 27.336)	24.349 (21.563 – 29.034)

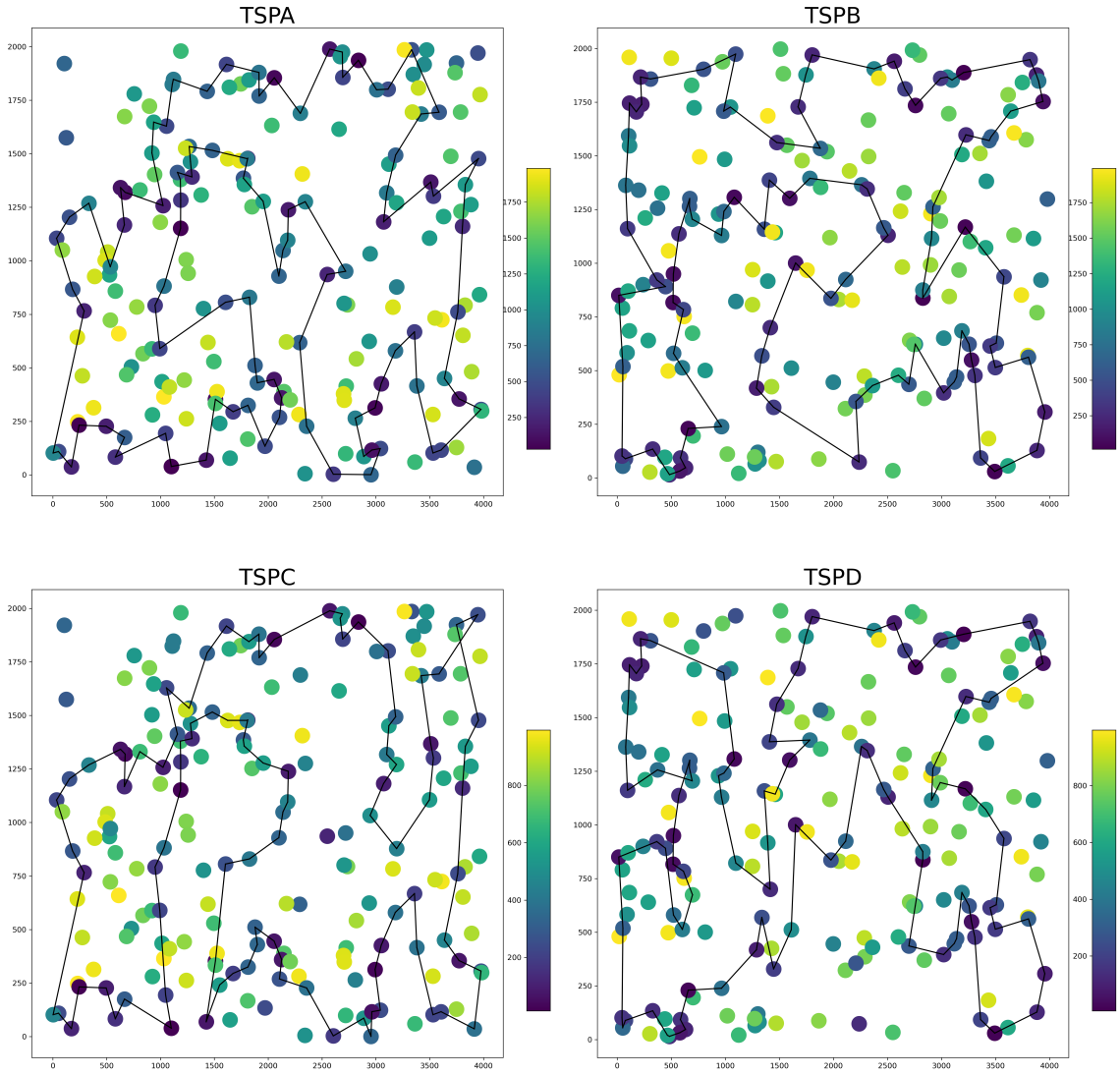
Best Solutions Plots

See plots: Plots

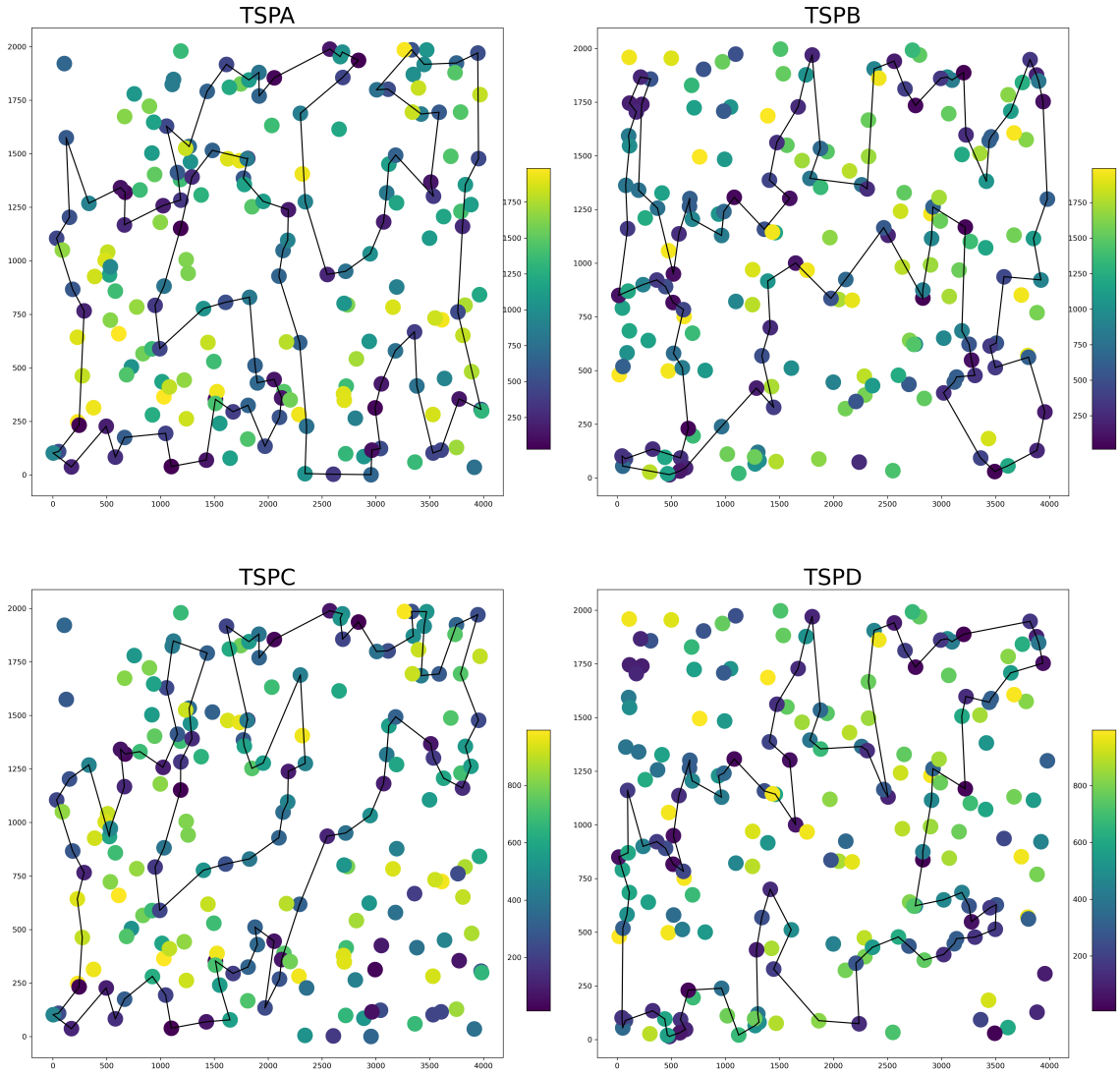
Greedy-inter-GreedyHeuristic



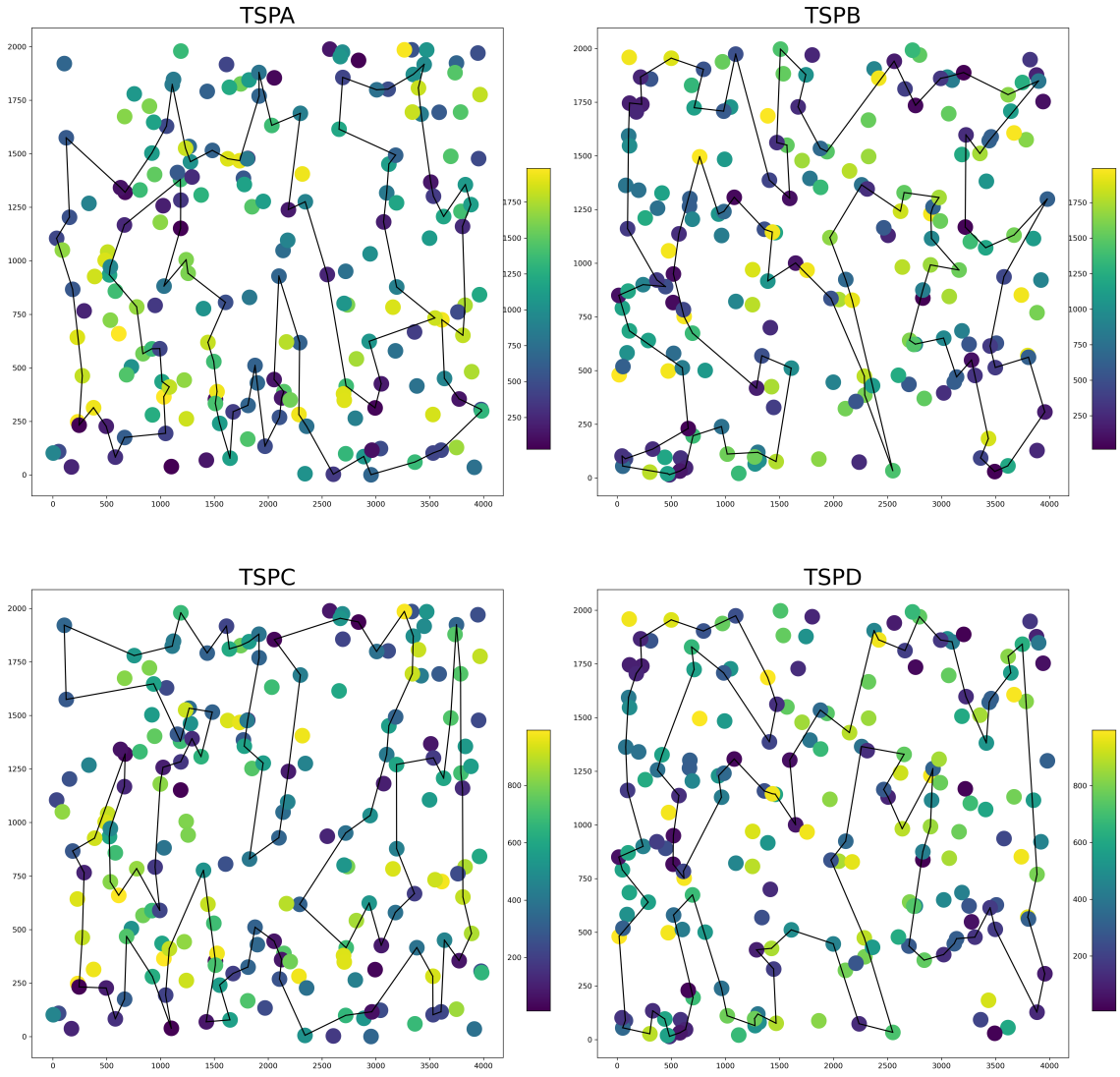
Greedy-inter-Random



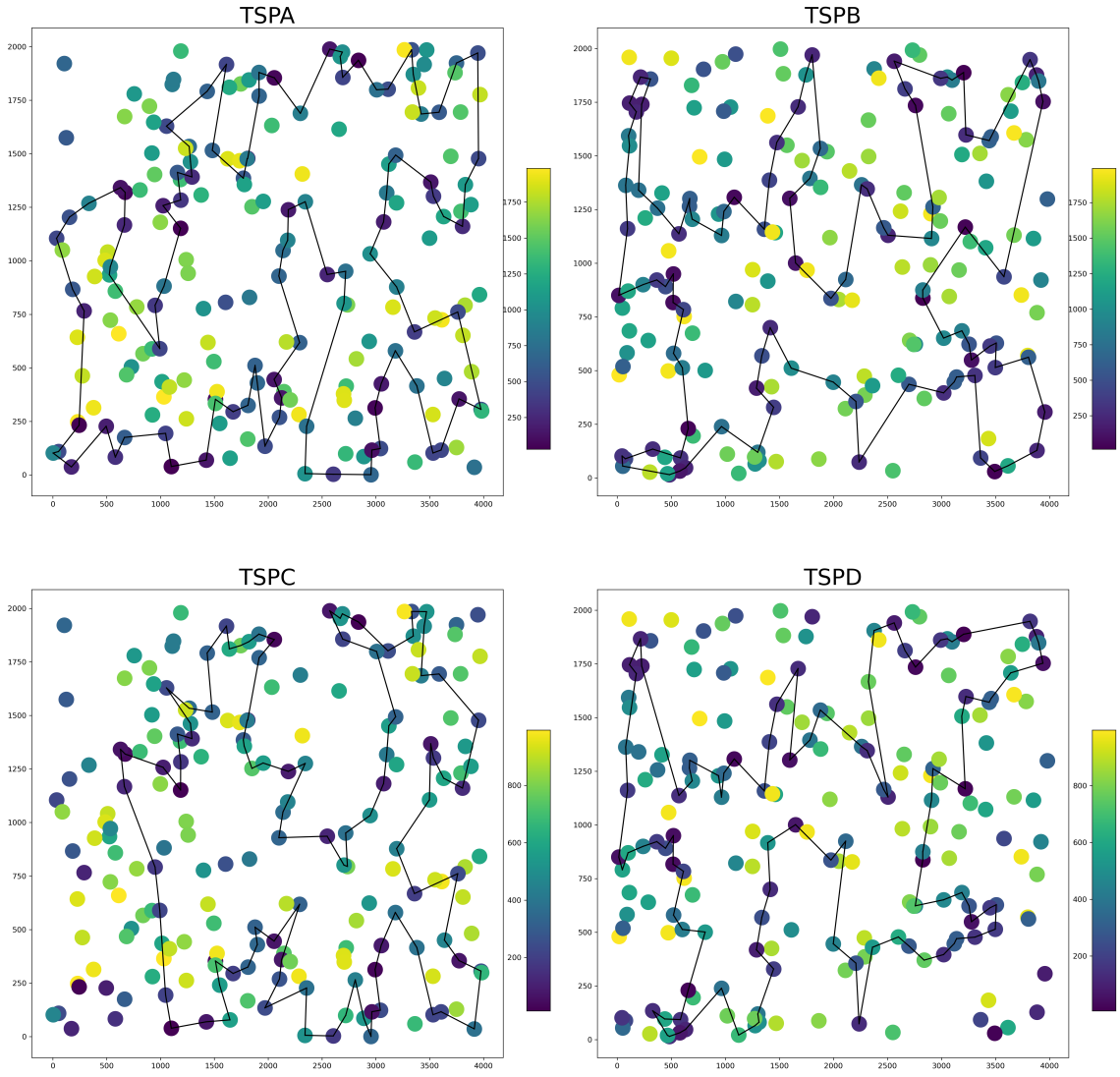
Greedy-intra-GreedyHeuristic



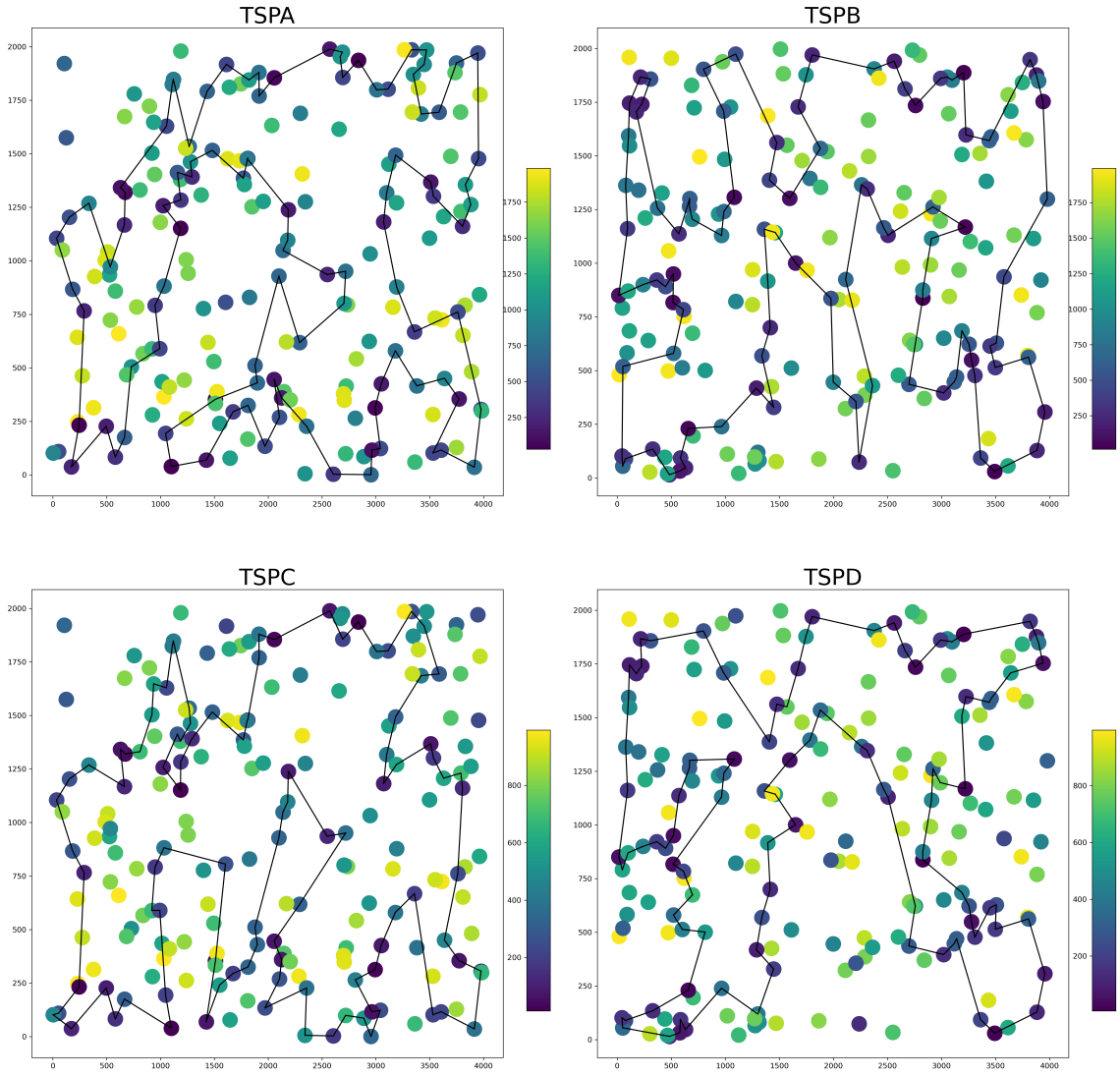
Greedy-intra-Random



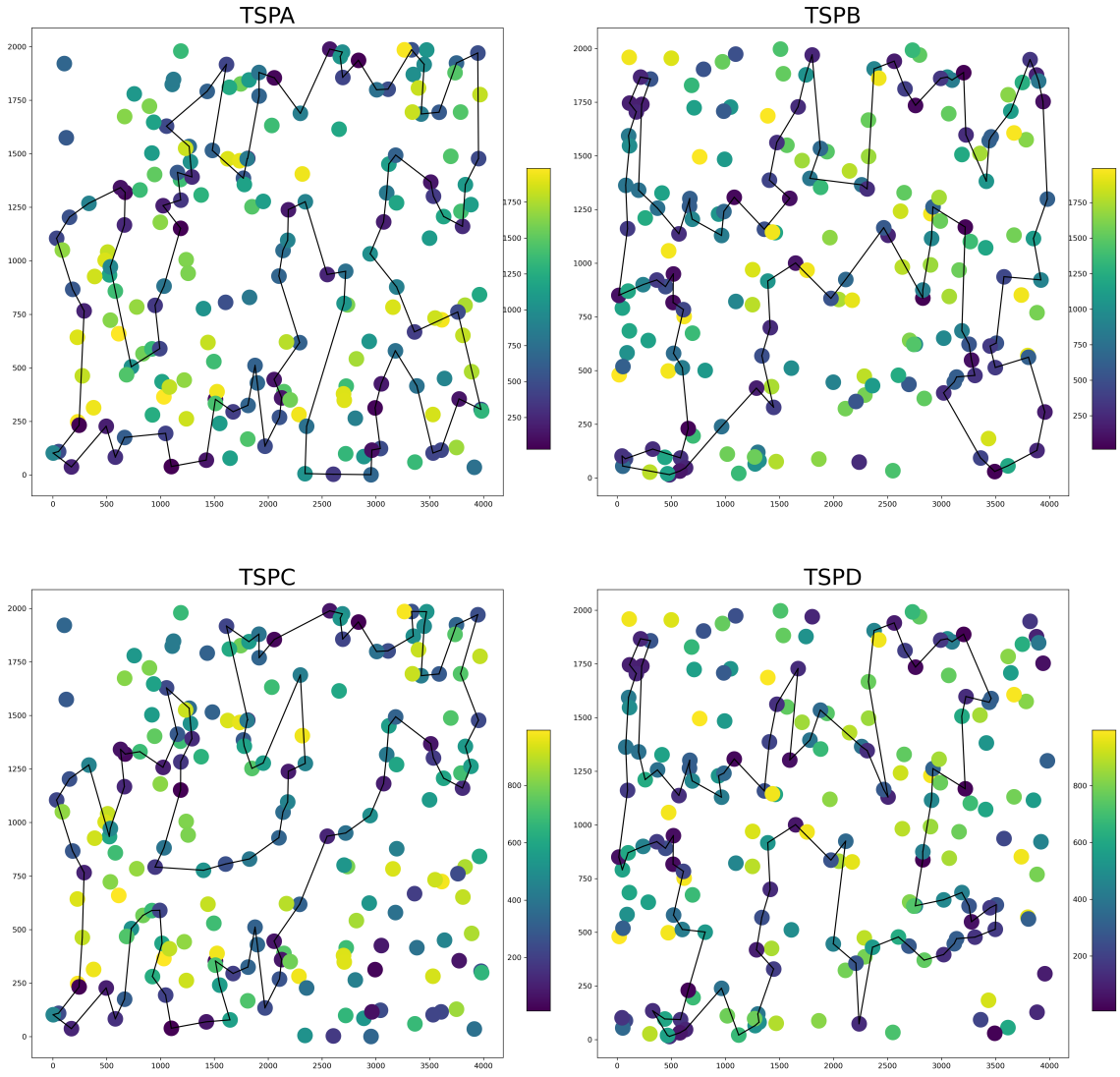
Steepest-inter-GreedyHeuristic



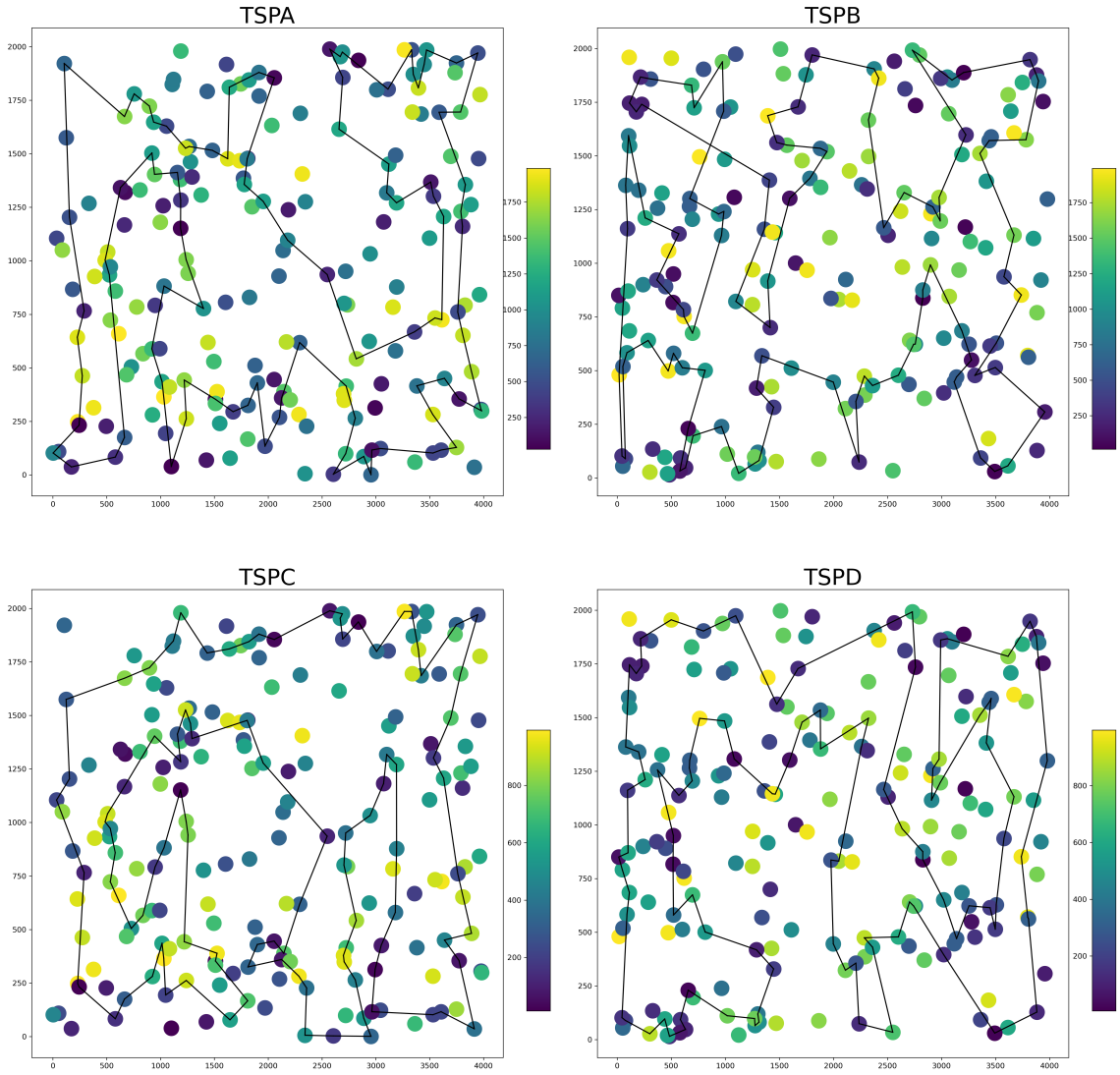
Steepest-inter-Random



Steepest-intra-GreedyHeuristic TSPA



Steepest-intra-Random



Conclusions

Performance

Greedy vs. Steepest

The Steepest approach generally resulted in lower total costs compared to the Greedy approach. This indicates that Steepest, which considers all possible neighbor moves before making a decision, tends to find better solutions at the expense of higher computational time.

Among the Steepest and Greedy approaches, the ones initialized with a Greedy Heuristic (Greedy-intra-GreedyHeuristic, Greedy-inter-GreedyHeuristic, Steepest-intra-GreedyHeuristic, Steepest-inter-GreedyHeuristic) typically performed better than their counterparts started with a Random approach, indicating the importance of a good starting solution.

Heuristic vs. Random Initial Solutions

Algorithms initialized with Greedy Heuristics outperformed those with Random initial solutions in terms of total cost, emphasizing the impact of a good starting point.

The consistency in solution quality was also better for algorithms that began with a Greedy Heuristic, suggesting that a good initial solution not only improves performance but also stability.

Computational Time

Greedy vs. Steepest Approaches

Greedy methods were faster but did not always provide solutions as good as those from the Steepest methods.

The latter require more time because they perform a more exhaustive search within the neighborhood before making a move.

The increased time for Steepest Descent methods is justified by the improved solution quality, making them suitable for scenarios where the quality of the solution is paramount and time is a secondary concern.

Heuristic vs. Random Initial Solutions

Starting with Greedy Heuristics generally led to faster convergence compared to Random initial solutions.

This is likely because Greedy Heuristics begin the search closer to good local optima, reducing the number of iterations needed.

Intra vs. Inter Neighborhoods

Performance

Intra-neighborhood approaches (where the moves are restricted within the current solution) and inter-neighborhood approaches (where the moves can include swapping with elements outside the current

solution) have shown competitive performances.

However, the inter-neighborhood methods tend to provide slightly better solutions, possibly due to a broader search space.

Computational Time

Inter-neighborhood searches generally took more time than intra-neighborhood searches.

This is expected because inter-neighborhood searches have a larger search space to explore, thus requiring more computational effort.