Michał Kałmucki 151944

Dominik Ludwiczak 151948

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.

Algorithm:

```
1. Function generateSolution(solution):
    a. Initialize ListOfMoves as an empty sorted set of Move objects
    b. Call createStartingList(solution)
    c. Initialize a SolutionChecker (distanceMatrix, nodeCosts)

    d. While true (loop indefinitely):
        i. Set foundBetterSolution = false
        ii. Set chosen = null

        iii. For each move (pm) in ListOfMoves:
            - If pm is applicable to solution:
                - Set foundBetterSolution = true
                - Apply pm to solution
                - Set chosen = pm
                - Break the loop

            - Else if pm is reverse applicable to solution:
                - reverse pm
                - Set foundBetterSolution = true
                - Apply pm to solution
                - Set chosen = pm
                - Break the loop

        iv. If foundBetterSolution is true:
            - Call deleteMoves
            - Call createNewMoves
        v. Else (no better solution found):
            - Break the loop

    e. Return the updated solution
```

```
2. Function deleteMoves(removedEdge1, removedEdge2):
    a. Initialize an empty list toRemove
    b. For each pm in ListOfMoves:
        i. If removedEdge1 or removedEdge2 is part of pm's edges, add pm to
toRemove
    c. Remove all moves in toRemove from ListOfMoves
```

```
3. Function createNewMoves(solution, createdEdge1, createdEdge2):
    a. For each node in solution:
        i. If node and following node are not adjecent to createdEdges create
ExchangeEdges moves between created edges and node
        ii. Add moves  to ListOfMoves if the objective change is less than 0

    b. For each node in solution involving one of new createdEdges:
        i. Create all possible node moves to nodes outside solution.
        ii. If the objective change is less than 0, add the new move to
ListOfMoves
```

```
4. Function createStartingList(solution):
    a. For each pair of nodes (i, j) in the solution:
        i. If nodes i and j are not adjacent, generate possible moves involving
them
        ii. Add these moves to ListOfMoves if the objective change is less than 0

    b. For each node in the solution:
        i. Generate potential new moves involving other nodes outside solution
        ii. If the objective change is less than 0, add the move to ListOfMoves
```

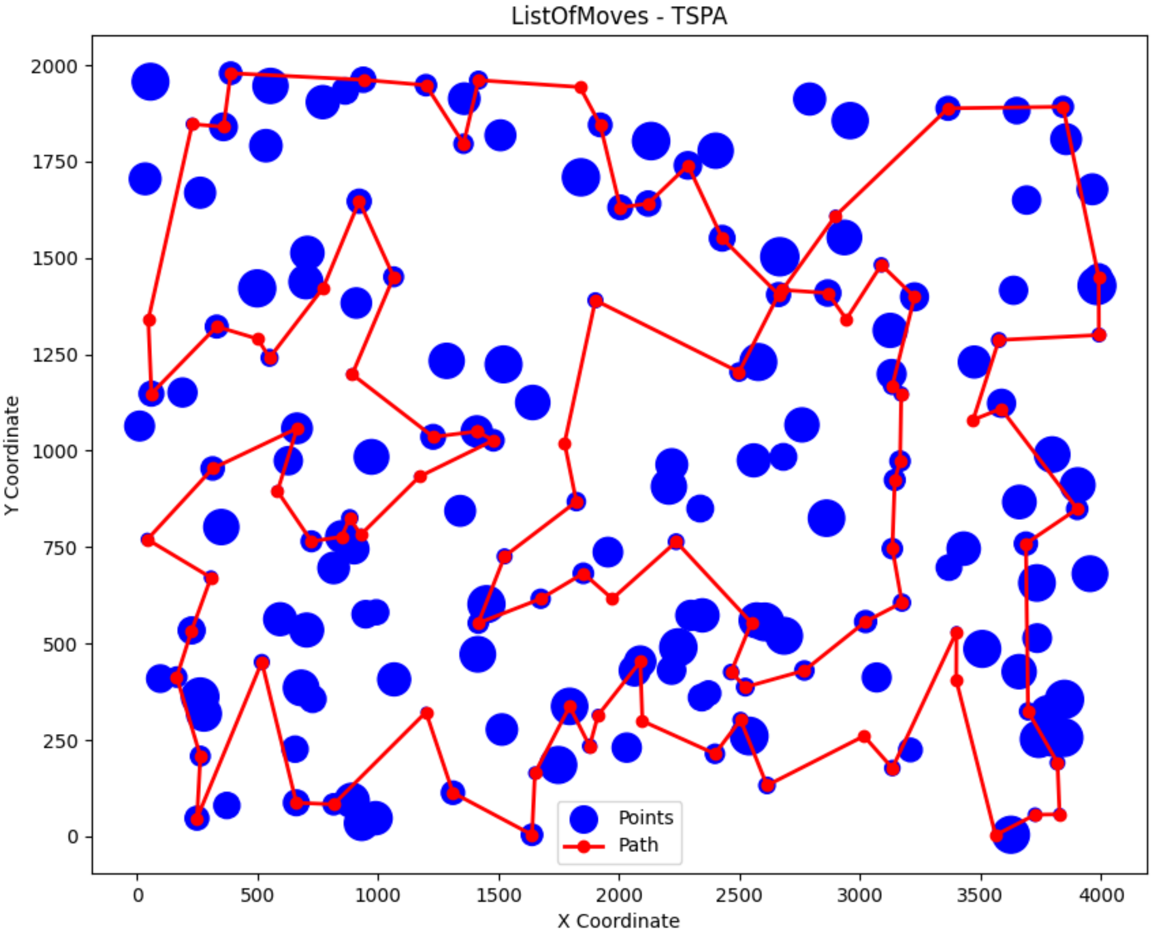
Results:

	Method	**Instance**	**Min Distance**	**Max Distance**	**Average Distance**	**Execution Time (ms)**
STANDARD STEEPEST	STEEPEST-EDGES-RANDOM	TSPA	71756	78077	74003.145	86739
		TSPB	46019	51211	48491.05	96498
	STEEPEST-NODES-RANDOM	TSPA	78755	96702	88257.14	108331
		TSPB	55195	73086	62964.445	110889

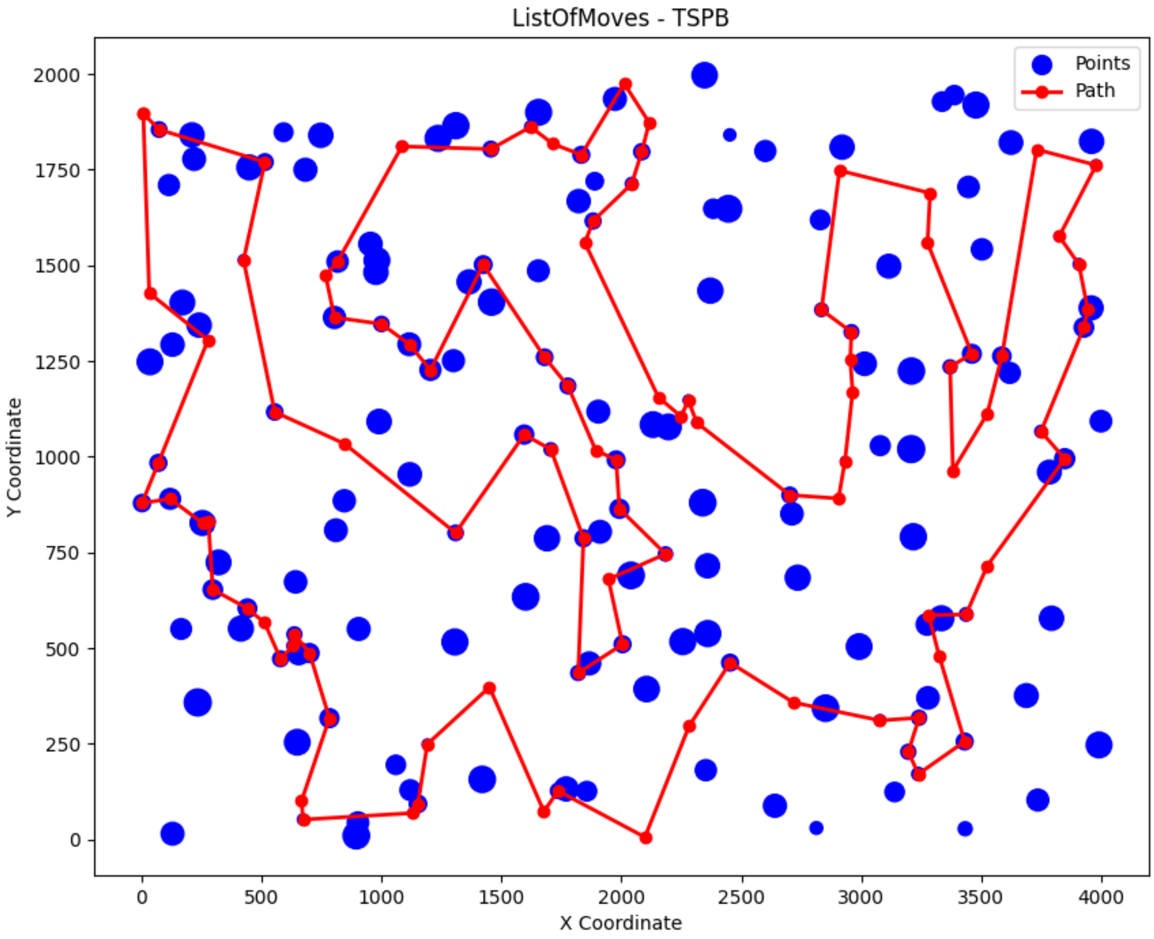
	Method	Instance	Min Distance	Max Distance	Average Distance	Execution Time (ms)
CANDIDATE EDGES	STEEPEST-CANDIDATE-RANDOM	TSPA	74279	84338	78094.115	10246
		TSPB	45854	52213	49281.485	9010
List Of Moves	LIST-OF-MOVES	TSPA	73373	82414	77467.08	9131
		TSPB	45695	52010	49133.28	7613

Graphs:

TSPA:



TSPB:


ListOfMoves - TSPB
Points
Path
Y Coordinate
X Coordinate
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500
4000