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. Initialize variables:
   - Set `foundBetterSolution` to `true`.

2. While a better solution is found:
   - Reset `minObjectiveChange` and `bestMove`.

3. For each element `i` in the solution:
   - Find 10 nearest neighbors:
     - compute 10 nearest neighbors using distance matrix and node costs.

   - Evaluate possible moves:
     - For each neighbor `j`:
       - If `j` is in the solution, use `EXCHANGE_EDGES`.
       - Otherwise, use `CANDIDATE_EDGE`.
     - Simulate the move and calculate objective change.

   - Select the best move:
     - Update `bestMove` if a better move is found.

4. Stop when no improvement is found:
   - If no `bestMove` is found, stop the search.
   - Otherwise, apply the `bestMove` and repeat.
```

```
Calculate `objectiveChange` as follows:
    previous:
        Remove distances:
            - distance from node1 to prevNode1
            - distance from prevNode1 to prevPrevNode1
            - nodeCost of prevNode1
```

```
        Add distances:
            - distance from node1 to node2
            - distance from node2 to prevPrevNode1
            - nodeCost of node2
    next:
        Remove distances:
            - distance from node1 to nextNode1
            - distance from nextNode1 to nextNextNode1
            - nodeCost of nextNode1
        Add distances:
            - distance from node1 to node2
            - distance from node2 to nextNextNode1
            - nodeCost of node2
    return min(previous, next)
```

Results:

TSPA Instance

	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
CANDIDATE EDGES	STEEPEST-CANDIDATE-RANDOM	TSPB	74279	84338	78094.115	10246
	STEEPEST-CANDIDATE-RANDOM	TSPB	45854	52213	49281.485	18969

Source code link: Github

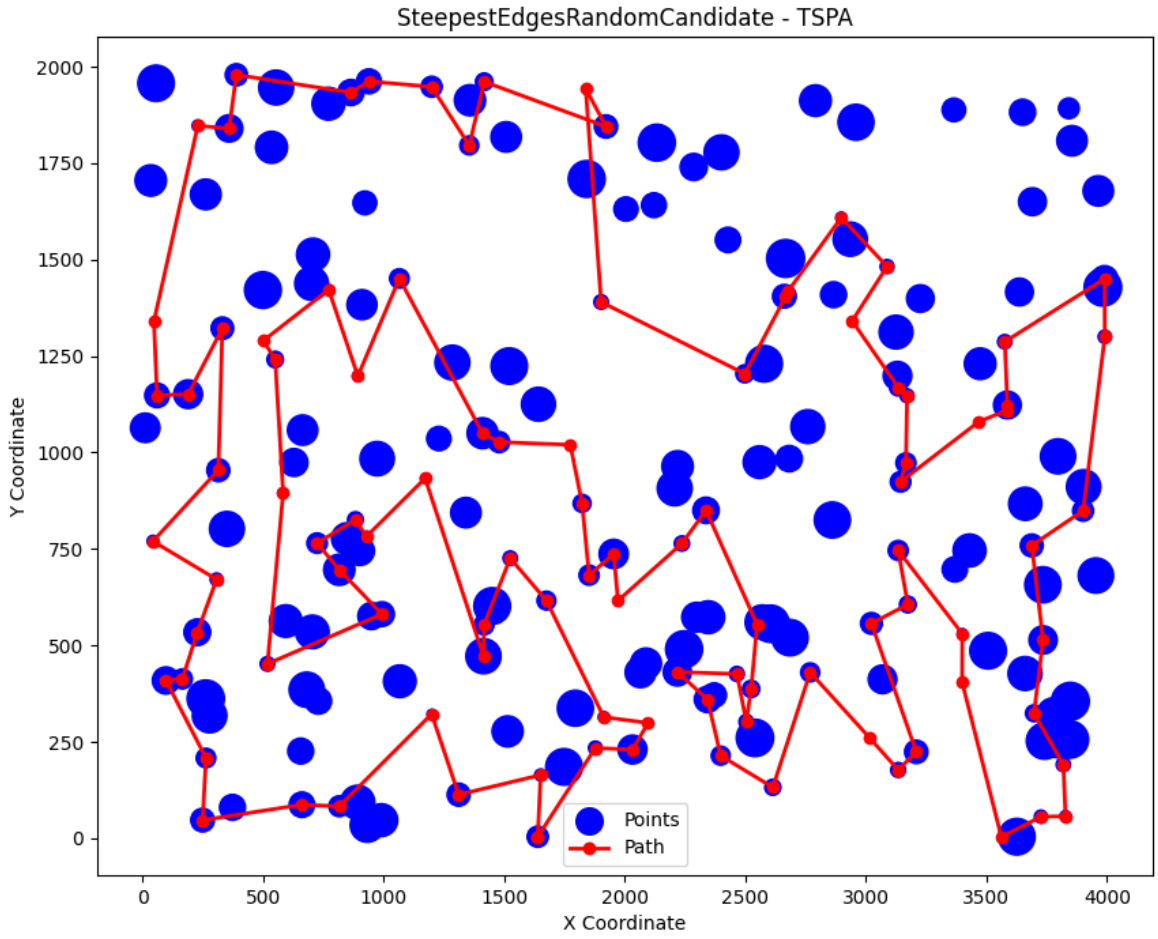
Conclusions:

Reducing the neighborhood to specific nearest candidate edges can lead to substantial improvements in computation time. While the differences in objective function values are noteworthy, it is crucial to prioritize a better neighborhood structure. Notably, exchanging nodes instead of edges resulted in even poorer outcomes compared to algorithms utilizing only nearest neighbors, while also incurring significantly higher

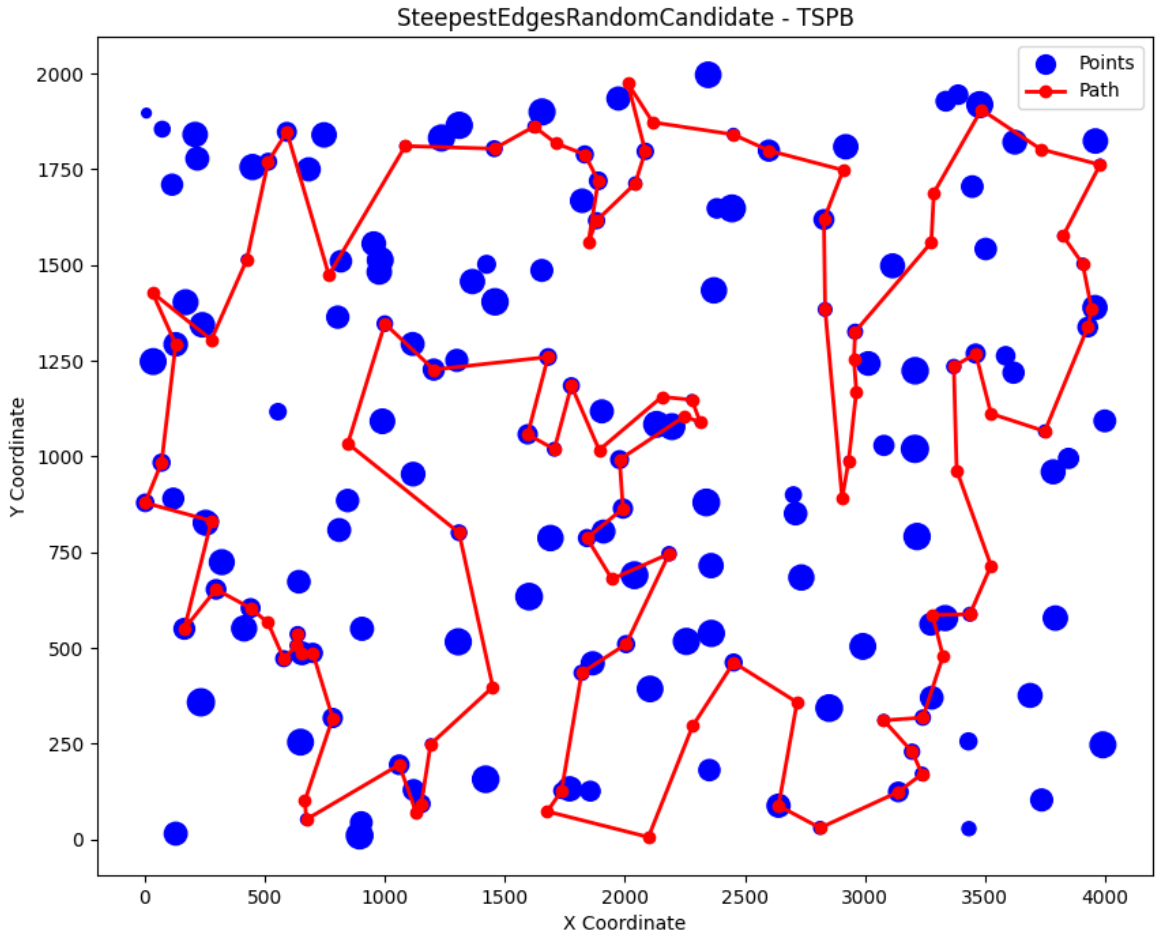
computational costs. This reinforces the importance of strategically selecting the types of moves in the search process to enhance both efficiency and solution quality.

Graphs:

TSPA:



TSPB:


SteepestEdgesRandomCandidate - 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