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.

PSEUDOCODE:

DESTROY:

```
Function Destroy(solution):
    Create a new list currSolution as a copy of solution
    Initialize objectiveChange to 0

    Repeat 3 times:
        Initialize nodes as an empty priority queue
        Initialize usedNodes as an empty list

        For 30 iterations:
            Select a random index from currSolution
            Get the corresponding node and ensure it is not in usedNodes
            Add the node and its associated cost to the nodes queue

        For 10 iterations:
            Remove the node with the highest priority from nodes
            Find the index of the node in currSolution
            Calculate the previous and next indices
            Update objectiveChange based on the removal of the node
            Remove the node from currSolution

    Return the modified currSolution and objectiveChange
```

REPAIR:

```
Function Repair(solution):
    Initialize takenNodes as a set containing elements of solution
    Initialize heuristic as WeightedRegret2
```

```
    Initialize objectiveChange as a list with a single element 0

    While the size of solution is less than half of the distanceMatrix size:
        Set solution to the result of calling FindRegret2 on heuristic with
solution and objectiveChange

    Return a Pair containing solution and the total objectiveChange
```

Results:

Method	Instance	Min Distance	Max Distance	Average Distance	Execution Time (ms)	Average Time (ms)	Average iterations
MSLS	TSPA	71615	78519	73886.605	2717553	16500.5	20
	TSPB	45111	51523	48358.664	1658511	18915.8	20
ILS	TSPA	69400	71285	70198.045	2004059	--	2890.785
	TSPB	43649	46946	44650.46	2003919	--	2964.39
LOCAL-SEARCH-STEEPEST	TSPA	71756	78077	74003.145	86739	--	--
	TSPB	46019	51211	48491.05	96498	--	--
GREEDY 2-REGRET WEIGHTED	TSPA	71,108	73,395	72,130.045	--	--	--
	TSPB	47,144	55,700	50,919.565	--	--	--
LNS	TSPA	69322	70801	70087.6	--	--	2459.15
	TSPB	43593	45139	44391.7	--	--	2598.25
LNS with LS	TSPA	69378	70572	69917.15	--	--	1487.3
	TSPB	43584	45047	44362.7	--	--	1539.4545

Source code link: Github

Conclusions:

Effectiveness in Solution Quality:

- The LNS (Large Neighborhood Search) method provides competitive results in terms of the solution quality. The average distance for TSPA is 70,087.6 and for TSPB is 44,391.7. These results are generally

close to the best values from other methods such as MSLS and ILS, indicating that LNS is effective in finding good solutions.
- When Local Search (LS) is applied to LNS, there is a slight improvement in solution quality. The LNS with LS method for TSPA has an average distance of 69,917.15, which is marginally better than the LNS without LS (70,087.6), suggesting that Local Search further fine-tunes the solution.

Graphs:

LNS without LS

- TSPA: a
- TSPB: b

LNS with LS

- TSPA: a
- TSPB: b