Authors: Weronika Zawadzka 151943 Eliza Czaplicka 151963

Date: November 24, 2024

Best solutions have been checked with the solution checker

Summary Report for w6_msls_ils

Based on methods from our github project

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 of implemented algorithms

- Multiple Start Local Search

Initiate LM – a set of moves that bring improvement ordered from the best to the worst, initially empty

Initaial best solution = Empty, Initial Best Cost = infinity

Run 200 times:

Generate an initial solution x (200 random, take best)

Initiate affectedNodes Set as initial solution

repeat until no move has been found after checking the whole list LM:

Evaluate all new moves based on affectedNodes and add improving moves to LM:

- Consider both intra (2 edges) and inter (2 nodes) exchanges

- Improving based on delta calculation - if delta < 0 then add to LM

- Considered are also moves with inverted edges

Browse moves from LM

newAffectedNodes = {};

for moves m from LM starting from the best:

check if move is valid/invalid or should be skipped

if valid then: apply, remove from LM, add affected nodes to newAffectedNodes break;

if invalid then remove from LM;

if should be skipped then skip;

affectedNodes = newAffectedNodes;

Calculate objective function. If lower than the best saved, save as the best.

CHECKING VALIDITY

1. Check if required nodes are still in the solution:

For inter-route moves: prevNode, node1, and nextNode must be in the solution. node2 must not be in the solution.

For intra-route moves: prevNode, node1, node2, and nextNode must all be in the solution. If any condition fails → INVALID (0).

2. Check if edges of interest between those nodes still in solution -> if not, invalid

For inter-route moves: Check adjacency: prevNode ↔ node1 node1 ↔ nextNode

For intra-route moves: prevNode ↔ node1 node2 ↔ nextNode If adjacency fails → INVALID (0).

3. Check if edges have same order -> if not, skip

For inter-route moves: Ensure prevNode, node1, and nextNode are in the correct relative order. If prevNode -> node1 -> nextNode is reversed → SKIP (2).

For intra-route moves: Ensure: prevNode -> node1 and node2 ->nextNode are in the correct order. Start index (node1) must be less than the end index (node2).

If the relative order is reversed → SKIP (2). 4. Else: Move is VALID (1).

- Iterated Local Search

Initiate LM – a set of moves that bring improvement ordered from the best to the worst, initially empty

Generate an initial solution x (200 random, take best)

Run local search (like in previous method) calculate objective function and save as current best

While runtime < set timeout:

Peturb current best solution

Run local search on peturbed solution

Calculate objactive function of generated solution from local search

If lower than current best then set as current best

PETURBATION

Randomly choose one of the three peturbations:

1. Randomly choose a segment (max 25%) of the solution and shuffle it

2. Randomly choose a segment (max 25%) of the solution and move it to a random place among the rest of the nodes

3. Randomly split the solution into 4 parts and randomly reorder these parts

Summary performance of each method

Instance: A

Summary for Execution Time (ms)

method	min	max	mean
Steepest2edgesCandidate	12	49	15.01
Steepest2edgesMoveEvals	45	183	70.035

MultipleStartLocalSearch	12157	14328	13383.2
IteratedLocalSearch	13383	13400	13385.95

Summary for Objective Function Value

method	min	max	mean
IteratedLocalSearch	69265	70318	69702.1
MultipleStartLocalSearch	72053	73027	72616.2
Steepest2edgesMoveEvals	72800	78951	75485.68
Steepest2edgesCandidate	74540	84266	77731.47

Summary for Iteration Count

method	min	max	mean
IteratedLocalSearch	2823	3733	3349.85

Instance: B

Summary for Execution Time (ms)

method	min	max	mean
Steepest2edgesCandidate	13	48	15.67
Steepest2edgesMoveEvals	46	199	74.12
MultipleStartLocalSearch	12284	14056	13172.15
IteratedLocalSearch	13172	13175	13173.4

Summary for Objective Function Value

method	min	max	mean
IteratedLocalSearch	43550	44539	43985.9
Steepest2edgesCandidate	46036	52029	48474.77
MultipleStartLocalSearch	46123	47377	46813.0
Steepest2edgesMoveEvals	46217	52879	49734.125

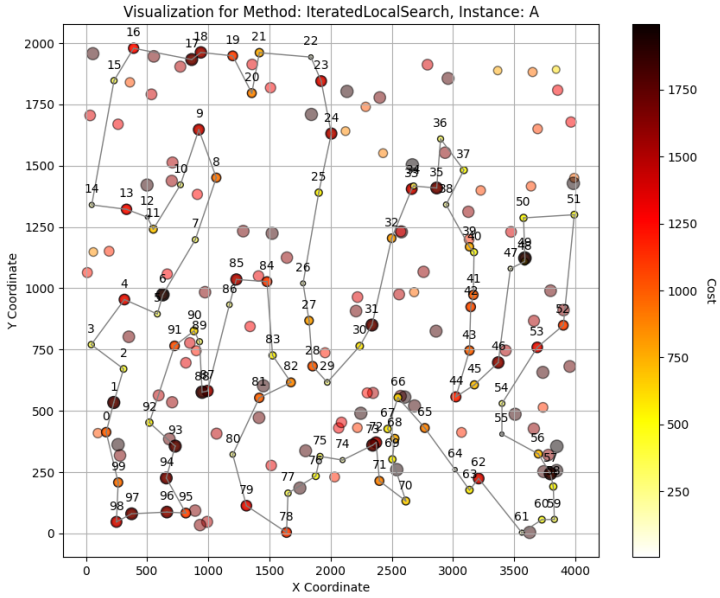
Summary for Iteration Count

method	min	max	mean
IteratedLocalSearch	2805	3594	3275.6

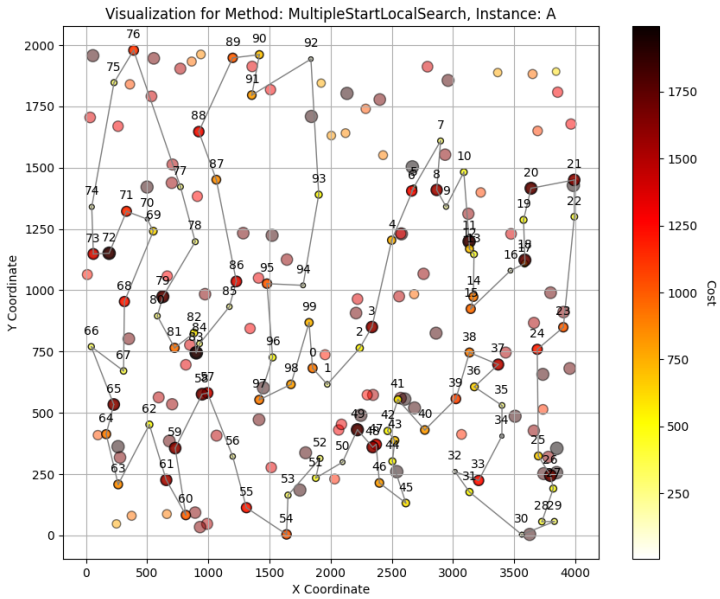
2D visualisations of best solutions

Instance: A

Method: IteratedLocalSearch with score=69265

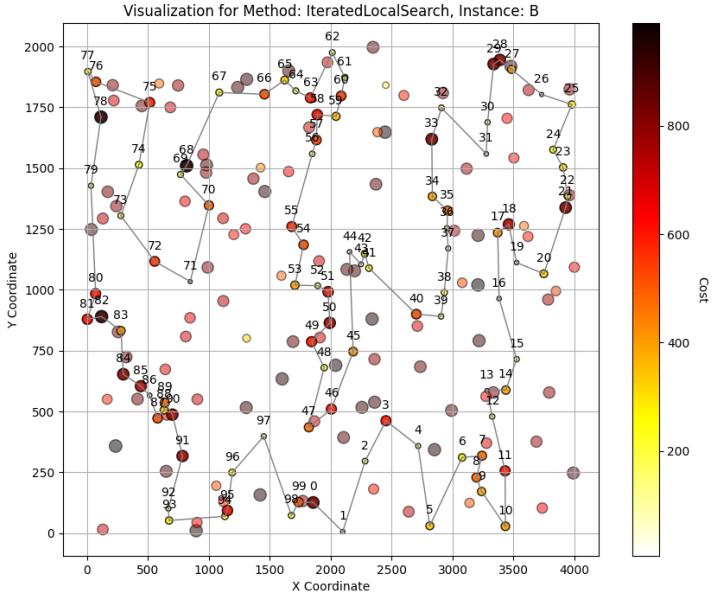


Method: MultipleStartLocalSearch with score=72053

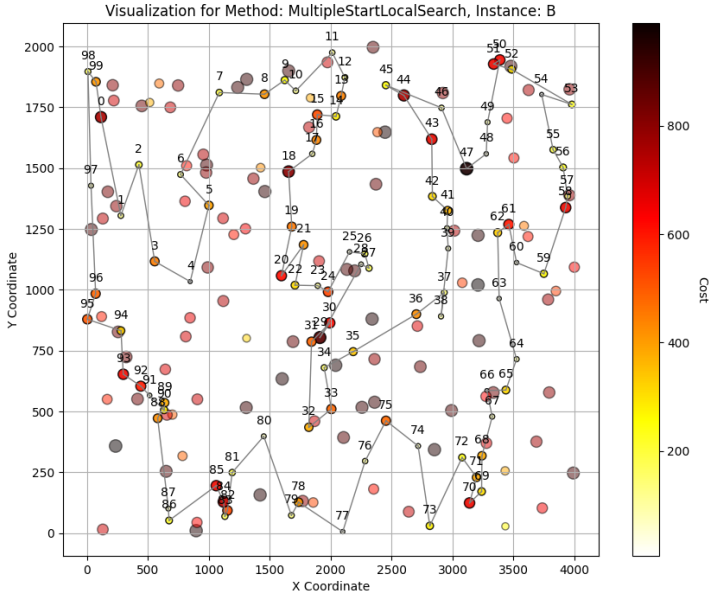


Instance: B

Method: IteratedLocalSearch with score=43550



Method: MultipleStartLocalSearch with score=46123



Best solutions, indices

Instance: A

Method: IteratedLocalSearch

Lowest Objective Function Value (f_val): 69265

Solution:

54, 48, 160, 34, 181, 42, 5, 115, 46, 68, 139, 41, 193, 159, 22, 18, 108, 140, 93, 117, 0, 143, 183, 89, 23, 137, 176, 80, 79, 63, 94, 124, 148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 57, 129, 92, 179, 185, 40, 119, 165, 90, 81, 196, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 2, 152, 97, 1, 101, 75, 86, 26, 100, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 149, 131, 65, 116, 43, 184, 35, 84, 112, 4, 190, 10, 177

Method: MultipleStartLocalSearch

Lowest Objective Function Value (f_val): 72053

Solution:

79, 63, 94, 124, 148, 62, 9, 144, 102, 49, 14, 3, 178, 106, 52, 55, 185, 40, 119, 165, 39, 27, 90, 81, 196, 31, 56, 113, 171, 175, 16, 44, 120, 25, 78, 145, 92, 179, 57, 129, 2, 152, 97, 1, 101, 75, 86, 26, 100, 121, 53, 154, 180, 135, 70, 127, 123, 149, 131, 35, 112, 84, 184, 177, 54, 48, 34, 160, 181, 41, 193, 159, 195, 146, 22, 18, 108, 139, 115, 5, 42, 43, 116, 47, 65, 59, 118, 46, 68, 117, 143, 0, 183, 137, 176, 51, 151, 162, 133, 80

Instance: B

Method: IteratedLocalSearch

Lowest Objective Function Value (f_val): 43550

Solution:

91, 141, 77, 81, 153, 187, 163, 103, 89, 127, 137, 114, 113, 176, 194, 166, 86, 95, 130, 185, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 143, 35, 109, 0, 29, 111, 82, 21, 8, 104, 144, 160, 33, 138, 11, 139, 168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6, 147, 191, 90, 51, 121, 131, 135, 122, 133, 107, 40, 100, 63, 38, 27, 16, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 175, 78, 5, 177, 36, 61

Method: MultipleStartLocalSearch

Lowest Objective Function Value (f_val): 46123

Solution:

100, 135, 122, 131, 121, 51, 90, 147, 6, 188, 169, 70, 3, 15, 145, 13, 195, 168, 43, 139, 182, 11, 138, 33, 160, 29, 109, 35, 0, 56, 144, 104, 21, 82, 8, 111, 143, 124, 106, 62, 18, 55, 34, 170, 184, 155, 152, 174, 183, 140, 149, 4, 28, 60, 20, 148, 47, 94, 66, 179, 185, 130, 95, 86, 166, 194, 176, 113, 103, 127, 165, 89, 163, 187, 153, 81, 77, 141, 61, 36, 177, 5, 78, 175, 142, 45, 80, 190, 193, 54, 31, 117, 198, 156, 1, 27, 38, 63, 40, 107

Conclusions

Our experiments with Multiple start local search gave us better results than only one run on Steepest local search with move evaluations which could be expected. It proves that the algorithm can get stuck in local optima and diversifying starting solutions can help with that. However, this approach is in turn a lot more time consuming as it takes around 13 seconds for a run to complete. We can also see that it is not the most efficient way of diversifying solutions. The iterated local search provided better solutions than the multiple start algorithm while running for the exact same time. We were not surprised when mutations and perturbations of already good solutions provided a better starting point than taking solutions randomly. It also allowed the method to check exponentially more solutions, around 3000 while multiple start algorithm made only 200 checks.