POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

LABORATORY 10: OWN METHOD

Piotr Balewski, 156037
Lidia Wiśniewska, 156063

POZNAŃ 2025

1 Problem Description

The problem is to select exactly 50% of the nodes (from n available) and build a Hamiltonian cycle through the selected nodes. The objective is to minimize the sum of two components:

1. The total length of the Hamiltonian cycle (calculated as the sum of Euclidean distances between consecutive nodes in the cycle).
2. The total cost of the selected nodes (the sum of the cost attributes for each selected node).

All distances are rounded to the nearest integer.

2 Large Neighborhood Search Ultimate (LNSU)

The goal of this laboratory was to design and implement a customized metaheuristic, either as a novel approach or as a modification of previously implemented algorithms. We selected Large Neighborhood Search (LNS) with an additional Local Search (LS) inside the main loop. We applied three modifications:

1. **Improved initial solution**
   Instead of starting from a random solution, the initial solution is generated using a greedy regret-based *NearestAny* heuristic with regret weight $w = 0.5$.
2. **Adaptive destroy operators**
   Two destroy operators are employed:
   - *destroyRelated*, which removes elements based on geographic or structural similarity,
   - *destroyHeuristic*, which applies cost-based removal.

   Initially, both operators are selected with equal probability. During the algorithm execution, their selection probabilities are adaptively updated based on performance. Operators that lead to improved or globally best solutions receive higher weights, increasing their likelihood of being selected in subsequent iterations.
3. **Simulated Annealing acceptance criterion**
   To prevent premature convergence, the standard LNS acceptance rule that allows only improving solutions is replaced with a *Simulated Annealing* (SA) acceptance mechanism. Non-improving solutions may be accepted with a probability dependent on the objective value difference and the current temperature. The temperature is gradually decreased according to a geometric cooling schedule.

The time limit for the algorithm was set equal to the average execution time of the MSLS algorithm from lab 6.

2.1 Destroy Operators

Two destroy operators are used within the LNS framework to partially remove elements from the current solution.

Heuristic-based destroy - the first operator removes nodes based on edge costs computed as the sum of inter-node distance and node costs. Removal probabilities are biased towards edges with higher costs, increasing the likelihood of eliminating unfavorable parts of the solution. Additionally, with a fixed probability, a contiguous subpath is removed to allow larger structural changes. The remaining removals are performed in a scattered, probabilistic manner until the target removal fraction is reached.

```
procedure Destroy(solution, instance, removeFraction)

n = size(solution)
targetRemove = round(n * removeFraction)

remaining = copy(solution)
removed = empty set

// --- Compute edge costs and probabilities ---
for each edge (A = sol[i], B = sol[(i + 1) mod n]) do
    edgeCost[i] = dist(A, B) + cost(A) + cost(B)
    maxCost = max(edgeCost)

for i in 0..n-1 do
    prob[i] = 0.2 + 0.8 * (edgeCost[i] / maxCost)

// --- With 30% probability remove a contiguous subpath ---
if random() < 0.30 then
    start = random integer in [0, n-1]
    len = random integer in [targetRemove / 3, targetRemove]

    for k from 0 to len-1 while removed.size < targetRemove do
        idx = (start + k) mod n
        node = solution[idx]
        if node not in removed then
            add node to removed
            remove node from remaining

// --- Scattered probabilistic removal ---
while removed.size < targetRemove do
    i = random integer in [0, n-1]

    if random() < prob[i] then
        node = solution[(i + 1) mod n]
        if node not in removed then
            add node to removed
```

```
            remove node from remaining

return remaining
```

Related destroy - the second operator focuses on removing geographically related nodes. A random seed node is selected from the solution, and the closest nodes to this seed (based on the distance matrix) are iteratively removed. This operator promotes localized changes and helps explore alternative configurations within a specific region of the solution.

```
procedure DestroyRelated(solution, instance, removeFraction)

n = size(solution)
targetRemove = round(n * removeFraction)

remaining = copy(solution)
removed = empty set

// --- Select a random seed node ---
seedNode = solution[random integer in [0, n-1]]
add seedNode to removed
remove seedNode from remaining

// --- Remove nodes closest to the seed ---
candidates = copy(remaining)
sort candidates by increasing dist(seedNode, node)

for i from 0 to targetRemove - 2 while i < size(candidates) do
    node = candidates[i]
    add node to removed
    remove node from remaining

return remaining
```

2.2 Simulated Annealing

To reduce the risk of premature convergence, a Simulated Annealing acceptance criterion is incorporated into the LNS main loop. Instead of accepting only improving solutions, worse solutions may be accepted with a probability that depends on the objective value difference and the current temperature. This mechanism allows temporary deterioration of solution quality in order to escape local optima. The temperature is gradually decreased using a geometric cooling schedule.

```
procedure SimulatedAnnealingAcceptance(x, y, fx, fy, T)

delta = fy - fx

```

```
if delta < 0 or random() < exp(-delta / T) then
    x = y
    fx = fy

T = T * coolingRate

return x, fx, T
```

2.3 LNSU pseudocode

```
procedure LNSU(instance, removeFraction, bonusLS, timeLimit, startNode, T = 100.0,
     coolingRate = 0.9995)

startTime = currentTime()
runs = 0

x = RegretNearestAny(instance, startNode, k = 2, w = 0.5)
(x, fx) = LocalSearch(x, instance)

xBest = x
fBest = fx

// --- Destroy operator weights ---
weights[0] = 1.0      // heuristic-based destroy
weights[1] = 1.0      // related destroy

// ===== Main LNS loop =====
while currentTime() - startTime < timeLimit do

    // --- Select destroy operator ---
    p = weights[0] / (weights[0] + weights[1])
    if random() < p then
        destroyType = 0
        y = Destroy(x, instance, removeFraction)
    else
        destroyType = 1
        y = DestroyRelated(x, instance, removeFraction)

    // --- Repair phase ---
    (y, fy) = Repair(instance, y, k = 2, w = 0.5)
    (y, fy) = LocalSearch(y, instance)

    // --- Simulated Annealing acceptance ---
    delta = fy - fx
    if delta < 0 or random() < exp(-delta / T) then
        x = y
        fx = fy

```

```
        // --- Reward destroy operator ---
        if delta < 0 then
            weights[destroyType] += 1.0

        if fx < fBest then
            weights[destroyType] += 3.0
            xBest = x
            fBest = fx

    T = T * coolingRate
    runs = runs + 1

return (xBest, fBest, runs)
```

3 Experimental Results

LNSU was run 20 times per instance with a time limit equivalent to the average MSLS runtime. StartNode was picked randomly.

3.1 TSPA: LNSU

Results: min: 69202 max: 69520 avg: 69316
Best Path: [183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 185, 40, 165, 90, 81, 196, 179, 57, 129, 92, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 2, 152, 97, 1, 101, 75, 86, 26, 100, 121, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 42, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 181, 146, 22, 159, 193, 41, 139, 115, 46, 68, 69, 18, 108, 140, 93, 117, 0, 143]

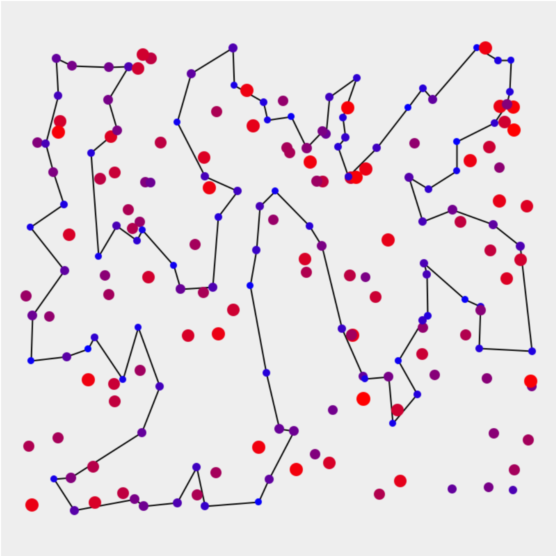

FIGURE 1: Best solution found by LNSU for TSPA.

3.2 TSPB: LNSU

Results: min: 43545 max: 44056 avg: 43853

Best Path: [168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6, 147, 90, 51, 121, 131, 135, 122, 133, 107, 40, 63, 38, 27, 16, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 162, 175, 78, 142, 45, 5, 177, 36, 61, 91, 141, 77, 81, 153, 187, 163, 89, 127, 103, 113, 176, 194, 166, 86, 185, 95, 130, 99, 22, 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, 8, 104, 144, 160, 33, 138, 11, 139]

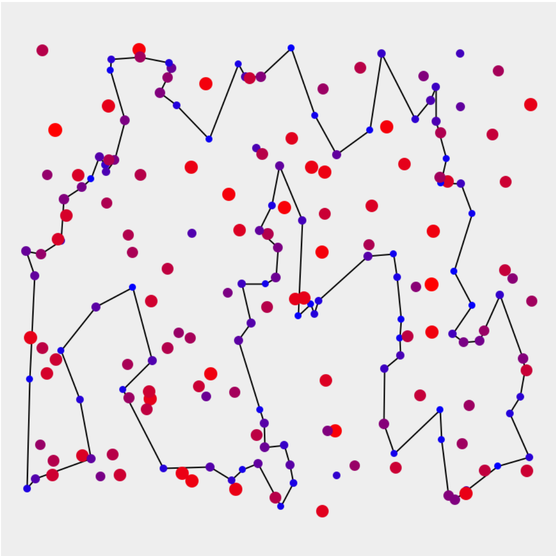

FIGURE 2: Best solution found by LNSU for TSPB.

Table 1: Results Summary

TSPA

Method	Avg	Min	Max
NNregretWeight ($w = 0.5$)	72401	70010	75452
Steepest Edge	75135	72495	79990
MSLS	72505	71431	73209
ILS	69462	69176	70196
LNS+LS	69454	69207	69865
LNSU	69316	69202	69520

TSPB

Method	Avg	Min	Max
NNregretWeight ($w = 0.5$)	47664	44891	55247
Steepest Edge	49275	46671	52022
MSLS	46766	46040	47280
ILS	43819	43535	44395
LNS+LS	43972	43636	44749
LNSU	43853	43545	44056

Table 2: Execution Times (ms)

TSPA

Method	Avg	Min	Max
NNregretWeight ($w = 0.5$)	54	43	127
Steepest Edge	48	40	173
MSLS	8903	8819	9226
ILS	8904	8903	8906
LNS+LS	8924	8905	8941
LNSU	8918	8905	8934

TSPB

Method	Avg	Min	Max
NNregretWeight ($w = 0.5$)	61	42	278
Steepest Edge	48	39	179
MSLS	8992	8873	9185
ILS	8993	8992	8994
LNS+LS	9011	8995	9051
LNSU	9009	8995	9066

Table 3: Number of Main Loop Iterations

TSPA

Method	Avg	Min	Max
ILS	5226	5037	5414
LNS+LS	279	235	294
LNSU	287	227	300

TSPB

Method	Avg	Min	Max
ILS	5471	5211	5667
LNS+LS	244	167	309
LNSU	290	252	302

4 Conclusions

The experimental results demonstrate that the proposed method consistently achieves high-quality solutions on both TSPA and TSPB instances, outperforming classical heuristics and matching or improving upon existing LNS variants.

Moreover, the adaptive mechanism and acceptance strategy allow LNSU to maintain solution quality while operating within the same computational time limits as other metaheuristics. The number of main loop iterations indicates that the algorithm effectively balances exploration and exploitation, confirming the benefits of the introduced modifications.

Solution Checker: All best solutions obtained were verified using the provided solution checker.
Source Code: https://github.com/PBalewski/EvolutionaryComputation/tree/main/lab10