POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

LABORATORY 7: LARGE NEIGHBORHOOD SEARCH

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

The Large Neighborhood Search (LNS) approach is designed to escape local optima by periodically destroying and repairing large portions of the current solution. Unlike classical Local Search—which modifies the solution only through small, incremental moves—LNS allows large structural changes, enabling the algorithm to explore distant regions of the search space while still exploiting local improvements.

The method operates by alternating between two main phases: *destroy* and *repair*.

```
procedure Run_LNS(instance, removeFraction, bonus_LS_iter, time_limit)

    start_time = getCurrentTime()

    init_sol = getRandomSolution(instance)
    (x, fx) = localSearch(init_sol, instance) // steepest LS intra_edge with LM
    clearMoveMemory()     // resets LM used in local search
    runs = 0

    while (getCurrentTime() - start_time) < time_limit do

        y = destroy(x, instance, removeFraction)

        (y, fy) = repair(instance, y, k = 2, regretWeight = 0.5)

        if bonus_LS_iter == true then
            (y, fy) = localSearch(y, instance)
            clearMoveMemory()

        if fy < fx then
            x = y
            fx = fy

        runs =+ 1

    return (x, fx, runs)
```

2.1 Destroy operator

The destroy operator removes a subset of nodes (removeFraction) from the current solution in order to create a partially destroyed solution for LNS. Removal is highly biased: edges with high cost (distance + node costs) are more likely to be broken. The destroy step first computes edge-based removal probabilities, then with 30% chance removes a contiguous subpath, and finally continues removing scattered nodes according to the biased probabilities until the desired number of removals 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
```

2.2 Repair operator

The repair operator reconstructs a complete cycle from the partially destroyed solution produced by the destroy step. It is based on the same greedy Nearest Neighbour (NearestAny) heuristic used in one of the previous assignments, with a fixed regret weight of 0.5. Unlike the standard version, which begins from a single starting node, the repair operator initializes the constructive heuristic with the set of nodes that remain after destruction. It then repeatedly inserts one of the nodes at the position yielding the smallest regret-weighted cost increase. This process continues until all removed nodes have been reinserted and a full cycle of length 100 is restored.

3 Experimental Results

The computational experiment consisted of evaluating the Large Neighborhood Search (LNS) algorithm both with and without an additional Local Search step. For each configuration, the algorithm was executed 20 independent runs on each of the two instances (TSPA and TSPB).

To ensure a fair comparison with previous work, the time limit for each LNS run was set equal to the average execution time of the MSLS algorithm (from the previous assignment) on the corresponding instance. This allowed both methods to operate under comparable computational budgets, enabling a meaningful assessment of solution quality and convergence behaviour.

3.1 TSPA: LNS

Results: min: 69273 max: 70154 avg: 69682
Best path: [183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 148, 9, 62, 144, 14, 49, 178, 106, 52, 55, 185, 40, 165, 39, 27, 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, 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]

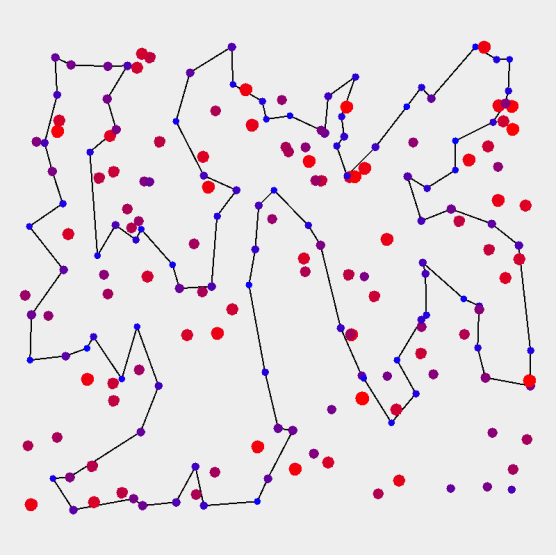

FIGURE 1: Best solution found by LNS for TSPA.

3.2 TSPA: LNS+LS

Results: min: 69207 max: 69865 avg: 69454

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

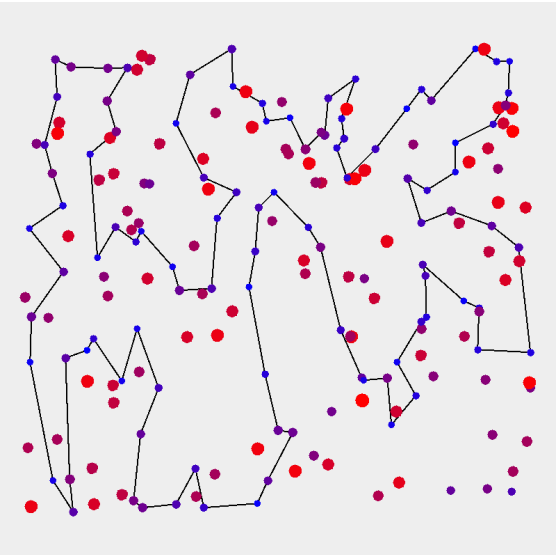

FIGURE 2: Best solution found by LNS+LS for TSPA.

3.3 TSPB: LNS

Results: min: 43630 max: 45268 avg: 44193

Best Path: [35, 109, 0, 29, 111, 82, 8, 104, 144, 160, 33, 138, 11, 139, 43, 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, 5, 177, 36, 61, 91, 141, 77, 81, 153, 187, 163, 103, 89, 127, 137, 114, 113, 176, 194, 166, 86, 185, 95, 130, 99, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 143]

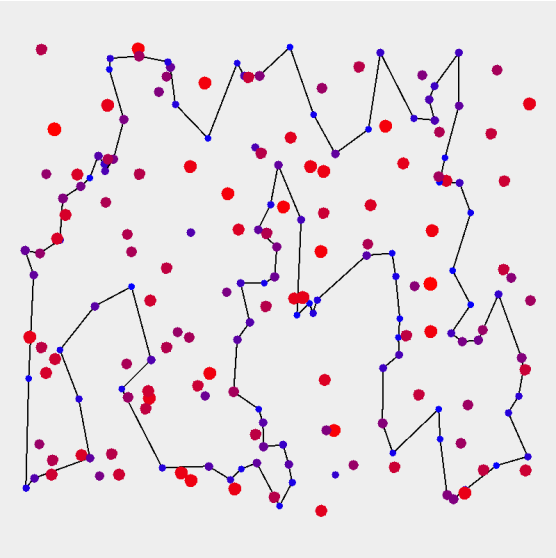

FIGURE 3: Best solution found by LNS for TSPB.

3.4 TSPB: LNS+LS

Results: min: 43636 max: 44749 avg: 43972

Best Path: [86, 185, 95, 130, 99, 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, 43, 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, 45, 142, 175, 78, 5, 177, 36, 61, 91, 141, 77, 81, 153, 187, 163, 89, 127, 103, 113, 176, 194, 166]

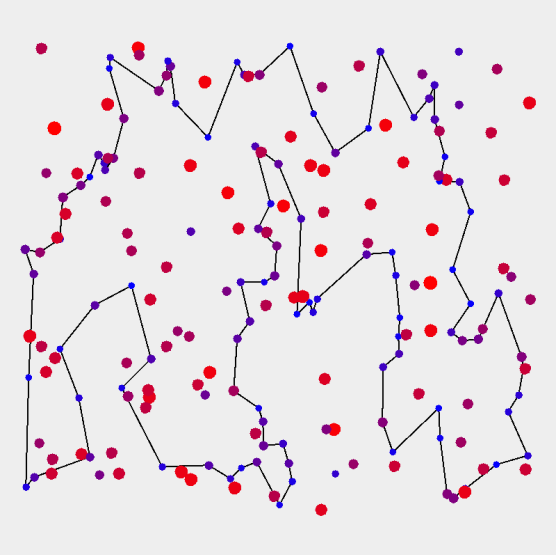

FIGURE 4: Best solution found by LNS+LS for TSPB.

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	69682	69273	70154
LNS+LS	69454	69207	69865

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	44193	43630	45268
LNS+LS	43972	43636	44749

Table 2: Execution times (in milliseconds) summary

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	8918	8905	8930
LNS+LS	8924	8905	8941

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	9015	8995	9070
LNS+LS	9011	8995	9051

Table 3: Nnumber of main loop iterations

TSPA

Method	Avg	Min	Max
ILS	5226	5037	5414
LNS	303	264	324
LNS+LS	279	235	294

TSPB

Method	Avg	Min	Max
ILS	5471	5211	5667
LNS	302	220	341
LNS+LS	244	167	309

4 Conclusions

The results show that Large Neighborhood Search (LNS) performs competitively compared to MSLS and ILS, achieving high-quality solutions despite running significantly fewer local search iterations. Adding a Local Search step inside LNS (LNS+LS) improves solution quality marginally but increases running time and reduces the number of main iterations performed. Interestingly, ILS remains the most consistent method, producing very stable results with minimal variability across runs. Overall, LNS offers a good trade-off between speed and quality. The experiments confirm that integrating destroy-and-repair mechanisms can effectively diversify the search compared to classical LS approaches.

Solution Checker: All best solutions obtained were verified using the provided solution checker.
Source Code: https://github.com/PBalewski/EvolutionaryComputation/tree/main/lab7