POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

LABORATORY 5: LOCAL SEARCH WITH USE OF DELTAS FROM PREVIOUS ITERATIONS

Piotr Balewski, 156037
Lidia Wiśniewska, 156063

POZNAŃ 2025

1 Problem Description

The problem description remains the same as in previous labs. 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 Steepest Local Search with Limited Memory (LS with Δ Reuse)

The goal of this assignment is to improve the time efficiency of the Steepest LS algorithm by implementing a Limited Memory (LM). This approach avoids the $O(N^2)$ or $O(N^3)$ computational cost of evaluating the entire neighborhood in every iteration. Instead, we maintain a list (LM) of known improving moves (deltas) and only re-evaluate moves locally around the most recently applied change.

2.1 Inter-route Moves (Node Exchange)

An inter-route move involves swapping a node already in the cycle i with a node outside of it k. The operation replaces i in the cycle with k, effectively modifying the connections between the predecessor and successor of i to include k instead. This allows exploration of new solutions by exchanging elements between the cycle and the set of unused nodes.

2.2 Intra-route Moves (Edge Exchange)

An intra-route move consists of reversing a subpath within the current cycle. This corresponds to exchanging two non-adjacent edges, effectively reconnecting the affected segment in the opposite direction. Such moves are useful for improving the internal structure of the cycle without changing the set of nodes included.

The full procedure looks following:

```
procedure steepestSearch(selected, remaining, inst, move_type)

LM = steepestSearch_LM
n = size(selected)

// --- 1. Initialize Local Move List if empty ---
if LM is empty then
    // --- Inter-route: Node Exchange ---
    for i_idx = 0 to n-1 do
        i_node = selected[i_idx]
        prev_i = selected[(i_idx - 1 + n) mod n]
        next_i = selected[(i_idx + 1) mod n]
        for each k_node in remaining do
            delta = calculateDeltaInterNode(selected, i_node, k_node, inst)
            if delta < 0 then
                add Move("inter", prev_i, next_i, i_node, k_node, delta) to LM

    // --- Intra-route: Edge Exchange ---
    for i_idx = 0 to n-1 do
        for j_idx = i_idx + 1 to n-1 do
            a = selected[i_idx]
            next_a = selected[(i_idx + 1) mod n]
            c = selected[j_idx]
            next_c = selected[(j_idx + 1) mod n]
            if next_c = a then continue
            delta = calculateDeltaIntraEdge(selected, a, c, inst)
            if delta < 0 then
                add Move("intra_edge", a, next_a, c, next_c, delta) to LM
end if

// --- 2. Recheck existing moves ---
sort LM by delta ascending
for each m in LM do
    (applyMove, keepMove) = validate(selected, remaining, m)
    if not keepMove then
        remove m from LM
        continue
    if not applyMove then
        continue

    // --- 3. Apply the move ---
    applyMove(selected, remaining, m)
    remove m from LM

    // --- 4. Update neighborhood ---
    changed = {m.A, m.B, m.C, m.D}
    addNewLocalMovesToLM(LM, selected, remaining, inst, changed, m.type)
    steepestSearch_LM = LM
    return true  // improvement found

return false  // no improving move found
end procedure
```

where validate function is:

```
procedure validate(succMap, remaining, m)
    switch m.type

        case "inter":
            acb = hasEdge(succMap, m.A, m.C) and hasEdge(succMap, m.C, m.B)
            bca = hasEdge(succMap, m.B, m.C) and hasEdge(succMap, m.C, m.A)
            d_in_remaining = (m.D in remaining)

            if ( (acb or bca) and d_in_remaining ) then
                return (true, true)    // valid and keep move
            else
                return (false, false)  // invalid, discard move
            end if

        case "intra_edge":
            ab = hasEdge(succMap, m.A, m.B)
            ba = hasEdge(succMap, m.B, m.A)
            cd = hasEdge(succMap, m.C, m.D)
            dc = hasEdge(succMap, m.D, m.C)

            // If either edge pair no longer exists -> invalid
            if ( (not ab and not ba) or (not cd and not dc) ) then
                return (false, false)
            end if

            // If both original or reversed edges still valid -> keep
            if ( (ab and cd) or (ba and dc) ) then
                return (true, true)
            end if

            // Otherwise: temporarily invalid but could remain for later
            return (false, true)

    end switch

    // Default: unknown move type
    return (false, false)
end procedure
```

and adding new moves function is:

```
procedure addNewLocalMovesToLM(LM, selected, remaining, inst, changedNodes,
    move_type)
    n = size(selected)

    // --- 1. Inter-route Re-evaluation ---
    indices_to_check_inter = {}
    for each node_id in changedNodes do
        idx = position of node_id in selected
        if idx != -1 then
            add (idx - 1) mod n to indices_to_check_inter
            add idx to indices_to_check_inter
            add (idx + 1) mod n to indices_to_check_inter
        end if
    end for

    // (A) Re-evaluate neighbors of affected positions with all remaining nodes
    for each i_idx in indices_to_check_inter do
        node_i = selected[i_idx]
        for each node_k in remaining do
            delta = calculateDeltaInterNode(selected, node_i, node_k, inst)
            if delta < 0 then
                prev = selected[(i_idx - 1 + n) mod n]
                next = selected[(i_idx + 1) mod n]
                add Move("inter", prev, next, node_i, node_k, delta) to LM
            end if
        end for
    end for

    // (B) If last move was "inter", re-evaluate possible reintroduction of the removed node
    if move_type = "inter" then
        node_c_old = first node in changedNodes such that node in remaining (or -1 if none)
        if node_c_old != -1 then
            for i_idx = 0 to n-1 do
                node_i = selected[i_idx]
                delta = calculateDeltaInterNode(selected, node_i, node_c_old, inst)
                if delta < 0 then
                    prev = selected[(i_idx - 1 + n) mod n]
                    next = selected[(i_idx + 1) mod n]
                    add Move("inter", prev, next, node_i, node_c_old, delta) to LM
                end if
            end for
        end if
    end if


    // --- 2. Intra-route Re-evaluation ---
    nodes_to_check = copy of changedNodes

    for each a_node in nodes_to_check do
```

```
        i_idx = position of a_node in selected
        if i_idx = -1 then continue

        for j_idx = 0 to n-1 do
            if i_idx = j_idx then continue

            // Ensure proper ordering
            if i_idx > j_idx then
                swap(i_idx, j_idx)
            end if

            // Skip adjacent edges
            if (j_idx = i_idx + 1) or (i_idx = 0 and j_idx = n - 1) then continue

            a = selected[i_idx]
            c = selected[j_idx]
            delta = calculateDeltaIntraEdge(selected, a, c, inst)

            if delta < 0 then
                next_a = selected[(i_idx + 1) mod n]
                next_c = selected[(j_idx + 1) mod n]
                add Move("intra_edge", a, next_a, c, next_c, delta) to LM
            end if
        end for
    end for

    // --- 3. Sort updated LM by best improvement ---
    sort LM by delta ascending
end procedure
```

3 Experimental Results

3.1 TSPA: steepestLM

Results: min: 71499 max: 78454 avg: 74278
Best path: [177, 10, 190, 4, 112, 84, 35, 131, 149, 47, 65, 116, 43, 42, 115, 59, 51, 176, 80, 79, 133, 151, 162, 123, 127, 70, 135, 154, 180, 53, 63, 94, 124, 152, 97, 1, 101, 86, 75, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 38, 157, 196, 81, 90, 165, 119, 40, 185, 179, 145, 78, 92, 57, 55, 52, 106, 178, 3, 49, 102, 14, 144, 62, 9, 148, 137, 23, 89, 183, 143, 0, 117, 93, 140, 68, 46, 139, 41, 193, 159, 69, 108, 18, 22, 181, 160, 34, 54, 48, 184]

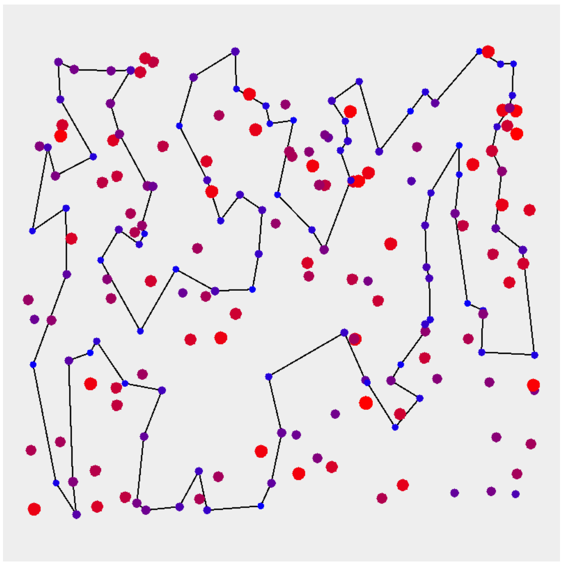
FIGURE 1: Best solution for steepestLM, TSPA.

3.2 TSPB: steepestLM

Results: min: 46213 max: 52112 avg: 48723

Best path: [18, 62, 128, 124, 106, 159, 143, 111, 8, 104, 138, 139, 11, 33, 160, 29, 0, 35, 109, 189, 155, 3, 70, 145, 168, 195, 13, 132, 169, 188, 6, 147, 178, 10, 133, 122, 90, 191, 51, 98, 118, 121, 131, 135, 107, 40, 63, 1, 24, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 175, 78, 5, 177, 21, 61, 36, 141, 77, 81, 153, 146, 187, 163, 89, 127, 103, 113, 180, 176, 194, 166, 86, 95, 130, 185, 179, 66, 94, 47, 148, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55]

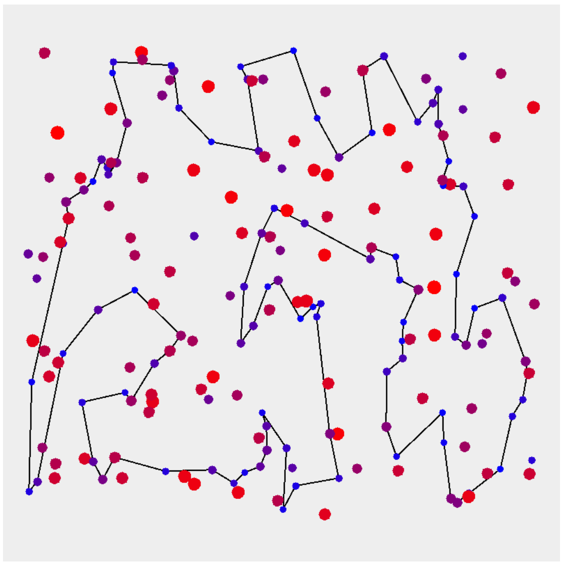
FIGURE 2: Best solution for steepestLM, TSPB.

Table 1: Results summary.

TSPA

Method	Average (Avg)	Min	Max
Lab 3: LS (Start: Random)			
R: Steepest+Node	88109	80805	95207
R: Steepest+Edge	75098	72746	79436
Lab 5: LS with LM (Start: Random)			
R: Steepest+Edge+LM	74278	71499	78454

TSPB

Method	Average (Avg)	Min	Max
Lab 3: LS (Start: Random)			
R: Steepest+Node	62972	55302	70406
R: Steepest+Edge	49275	46671	52022
Lab 5: LS with LM (Start: Random)			
R: Steepest+Edge+LM	48723	46213	52112

Table 2: Execution times (in milliseconds) summary.

TSPA

Method	Average (Avg)	Min	Max
Lab 3: LS (Start: Random)			
R: Steepest+Node	75,49	55,15	352,30
R: Steepest+Edge	55,12	44,83	115,60
Lab 5: LS with LM (Start: Random)			
R: Steepest+Edge+LM	42,90	30,78	218,94

TSPB

Method	Average (Avg)	Min	Max
Lab 3: LS (Start: Random)			
R: Steepest+Node	85,70	55,29	287,95
R: Steepest+Edge	65,75	45,59	187,54
Lab 5: LS with LM (Start: Random)			
R: Steepest+Edge+LM	44,79	27,35	207,19

4 Conclusions

Based on the results for TSPA and TSPB, the introduction of the Limited Memory (LM) strategy to the Steepest Local Search (LS) proved highly effective. The LS with LM consistently achieved better average and minimum solution costs compared to the standard full steepest search (R: Steepest+Edge) across both problem instances (TSPA: 74278 vs 75098; TSPB: 48723 vs 49275). This improvement suggests that the LM heuristic, by guiding the search with a focused list of high-impact moves, navigated the solution space more effectively, often leading to the discovery of deeper local optima. Crucially, the LM strategy successfully reduced the execution time, validating the main goal of the assignment. Average running time improved by approximately 22% for TSPA and 32% for TSPB.

Solution Checker: All best solutions obtained were verified using the provided solution checker.
Source Code: https://github.com/PBalewski/EvolutionaryComputation/tree/main/lab5