Evolutionary Computation
Local Search with LM Report

Authors and Source Code

- **Authors:**
  - Maksymilian Żmuda-Trzebiatowski 156 051
  - Krzysztof Bryszak 156 052

- **Source Code Repository:** https://github.com/MZmuda-Trzebiatowski/Evo_Comp_LS_LM

Problem Description

The problem involves a set of nodes, each defined by three columns of integers:

1. **X-coordinate**
2. **Y-coordinate**
3. **Node Cost**

The goal is to select exactly 50% of the nodes (rounding up if the total number of nodes is odd) and form a Hamiltonian cycle (a closed path) through the selected set. The objective is to minimize the total sum of the path length plus the total cost of the selected nodes.

- Distance Calculation: Distances are calculated as Euclidean distances, mathematically rounded to integer values.
- Optimization Constraint: A distance matrix must be calculated immediately after reading an instance. The optimization methods should only access this distance matrix, not the original node coordinates.

Implemented Algorithms (pseudocode)

- **Local Search Steepest, 2-opt with LM**

Input:
• tour: current cycle
• d: distance matrix
• nodes: node attributes (including cost)

Output:
• locally optimized tour

```
FUNCTION local_search_steepest_LM(tour, d, nodes):
    n ← size(d)
    k ← length(tour)

    FOR each v in tour:
        is_selected[v] ← true
    FOR i from 0 to k−1:
        node_pos[ tour[i] ] ← i

    LM ← empty min-priority-queue
    temp_invalid ← empty list
    affected_nodes ← set( tour )
    WHILE TRUE:
        // --- Refresh LM with moves involving affected nodes ---
        FOR each node in affected_nodes:
            update_LM(tour, d, nodes, is_selected, LM, node, node_pos)
        affected_nodes.clear()
        // Reinsert temporarily invalid moves from previous iteration
        FOR each move in temp_invalid:
            LM.push(move)
```

```
temp_invalid.clear()


applied ← FALSE
// === PROCESS MOVE LIST (Steepest-first) ===
WHILE LM not empty:
    m ← LM.pop()
    permanently_invalid ← FALSE
    applicable ← FALSE
    // variables for application
    apply_case ← 0
    ve_index ← −1
    i_idx ← −1
    j_idx ← −1
    //            VALIDATION & APPLICABILITY
    IF m.type = 0 THEN
        // --- V–E ---
        IF NOT is_selected[m.n2] THEN
            permanently_invalid ← TRUE
        ELSE IF is_selected[m.n4] THEN
            permanently_invalid ← TRUE
        ELSE
            idx ← node_pos[m.n2]
            IF tour[idx−1] = m.n1 AND tour[idx+1] = m.n3 THEN
                applicable ← TRUE
                apply_case ← 1
                ve_index ← idx
            ELSE
```

```
            temp_invalid.append(m)
    ELSE IF m.type = 2 THEN
        // --- 2-OPT ---
        IF NOT all four nodes (n1,n2,n3,n4) are selected THEN
            permanently_invalid ← TRUE
        ELSE
            idx ← node_pos[m.n1]
            jdx ← node_pos[m.n3]
            // forward-forward
            IF tour[idx+1] = m.n2 AND tour[jdx+1] = m.n4 THEN
                applicable ← TRUE
                apply_case ← 2
                i_idx ← idx
                j_idx ← jdx
            // reverse-reverse
            ELSE IF tour[idx−1] = m.n2 AND tour[jdx−1] = m.n4 THEN
                applicable ← TRUE
                apply_case ← 3
                i_idx ← idx−1
                j_idx ← jdx−1
            // broken edges → permanently invalid
            ELSE IF neither successor nor predecessor matches THEN:
                permanently_invalid ← TRUE
            ELSE
                temp_invalid.append(m)
    //                  APPLY MOVE
    IF applicable THEN
```

```
            IF apply_case = 1 THEN
                // V–E
                apply_V_E_exchange(tour, ve_index, m.n4)
                is_selected[m.n2] ← FALSE
                is_selected[m.n4] ← TRUE
                node_pos[m.n4] ← node_pos[m.n2]
                node_pos[m.n2] ← −1
            ELSE IF apply_case = 2 OR apply_case = 3 THEN
                // 2-OPT
                apply_2opt(tour, i_idx, j_idx)
                update_pos_after_2opt(tour, i_idx, j_idx, node_pos)
            affected_nodes.add( m.n1 )
            affected_nodes.add( m.n2 )
            affected_nodes.add( m.n3 )
            affected_nodes.add( m.n4 )
            applied ← TRUE
        // permanently invalid moves are simply discarded
    // === End LM scanning ===
    IF applied = FALSE THEN
        BREAK   // local optimum reached
  RETURN tour
END
```

Results and Analysis

Table of results

Algorithm	Instance A	Instance B
Random	264152 (239114 - 291474)	212540 (185581 - 238526)
NN end-only	85108.5 (83182 - 89433)	54390.4 (52319 - 59030)
NN all-pos	73302.4 (71695 - 75953)	48498.9 (44242 - 57283)
Greedy Cycle	72617.6 (71488 - 74410)	51339.5 (48765 - 57324)
NN all-pos 2-regret	117138 (108151 - 124921)	74444.5 (69933 - 80278)
Greedy Cycle 2-regret	115579 (105692 - 126951)	72740 (67809 - 78406)
NN all-pos 2-regret weighted (0.5, 0.5)	72401.2 (70010 - 75452)	47664.5 (44891 - 55247)
Greedy Cycle 2-regret weighted (0.5, 0.5)	72129.7 (71108 - 73395)	50897.1 (47144 - 55700)
Steepest LS swap, rand init	88179.1 (80805 - 97462)	62949.8 (54696 - 71421)
Steepest LS swap, greedy init	72010 (69801 - 75440)	47137 (44488 - 54391)
Steepest LS 2-opt, rand init	73975.5 (71248 - 78900)	48421.5 (45882 - 51676)
Steepest LS 2-opt, greedy init	70722.3 (69540 - 72546)	46342 (44320 - 51431)
Greedy LS swap, rand init	86548 (79976 - 94362)	61330.1 (54462 - 70020)
Greedy LS swap, greedy init	72010.4 (69801 - 75440)	47108.3 (44456 - 54372)
Greedy LS 2-opt, rand init	73324.9 (71455 - 76688)	48189.2 (44632 - 51038)
Greedy LS 2-opt, greedy init	70943.8 (69497 - 73149)	46372.4 (44320 - 51462)
Steepest LS 2-opt, rand init, candidate	77709.6 (73310 - 82396)	48362.6 (45822 - 52155)
Steepest LS 2-opt, rand init with LM	75152.7 (72247 - 80243)	49606.8 (46672 - 52878)

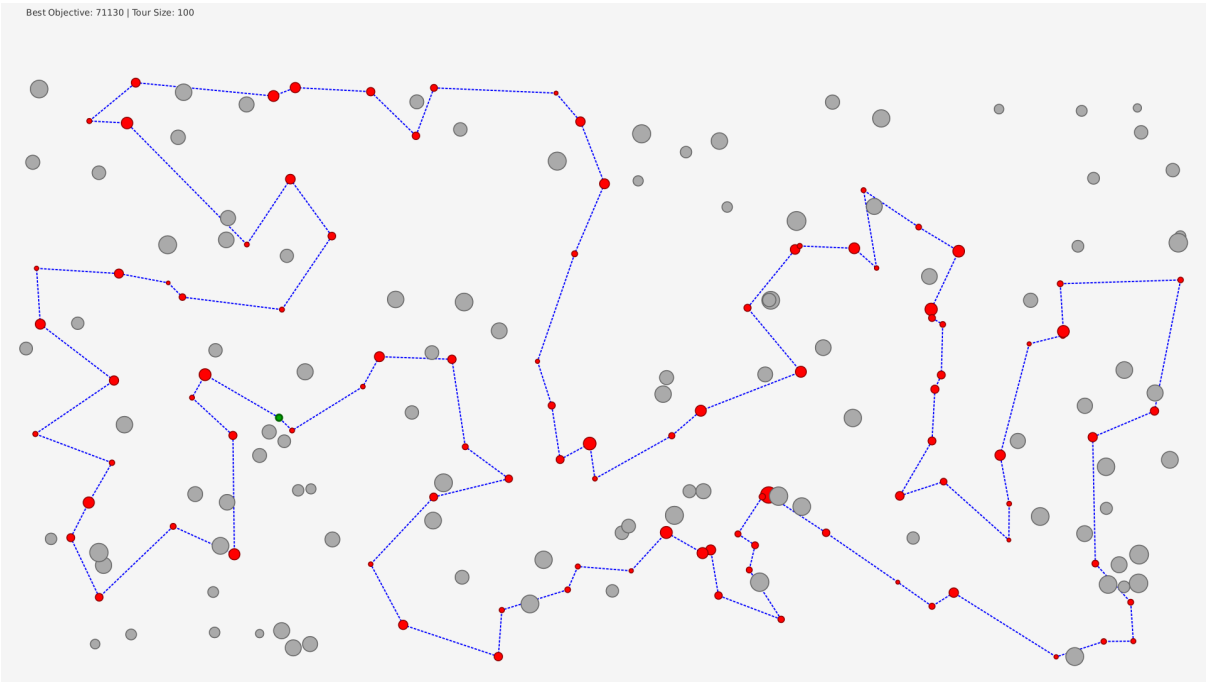
Table of Runtimes

Algorithm	Instance A (runtime in sec)	Instance B (runtime in sec)
Steepest LS swap, rand init	1.964	1.932
Steepest LS swap, greedy init	1.245	0.297
Steepest LS 2-opt, rand init	1.348	1.344
Steepest LS 2-opt, rand init cand + list	0.711	0.663
Steepest LS 2-opt, rand init with LM	0.248	0.26
Steepest LS 2-opt, greedy init	0.28	0.318
Greedy LS swap, rand init	0.44	0.33
Greedy LS swap, greedy init	0.236	0.245
Greedy LS 2-opt, rand init	0.362	0.27
Greedy LS 2-opt, greedy init	0.259	0.254

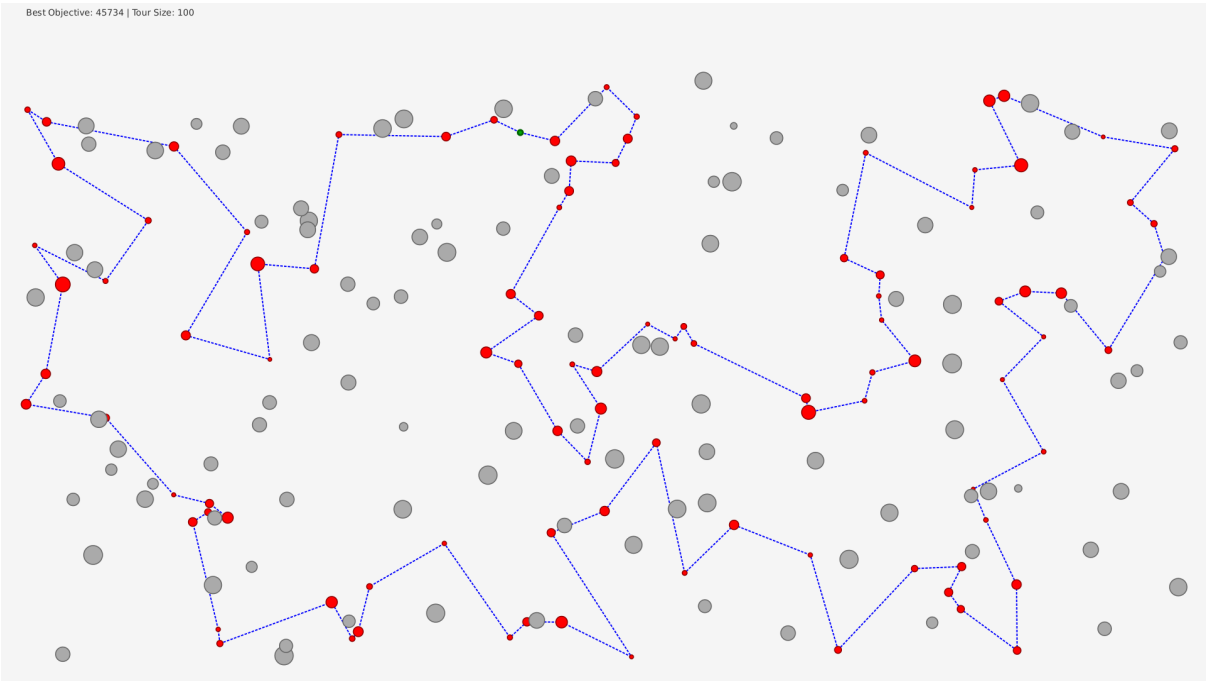
Visual Comparisons (Visual Comparision) - Instance A

The size of the dot corresponds to its cost (the bigger it is the bigger the cost), and the green dot is the starting node.

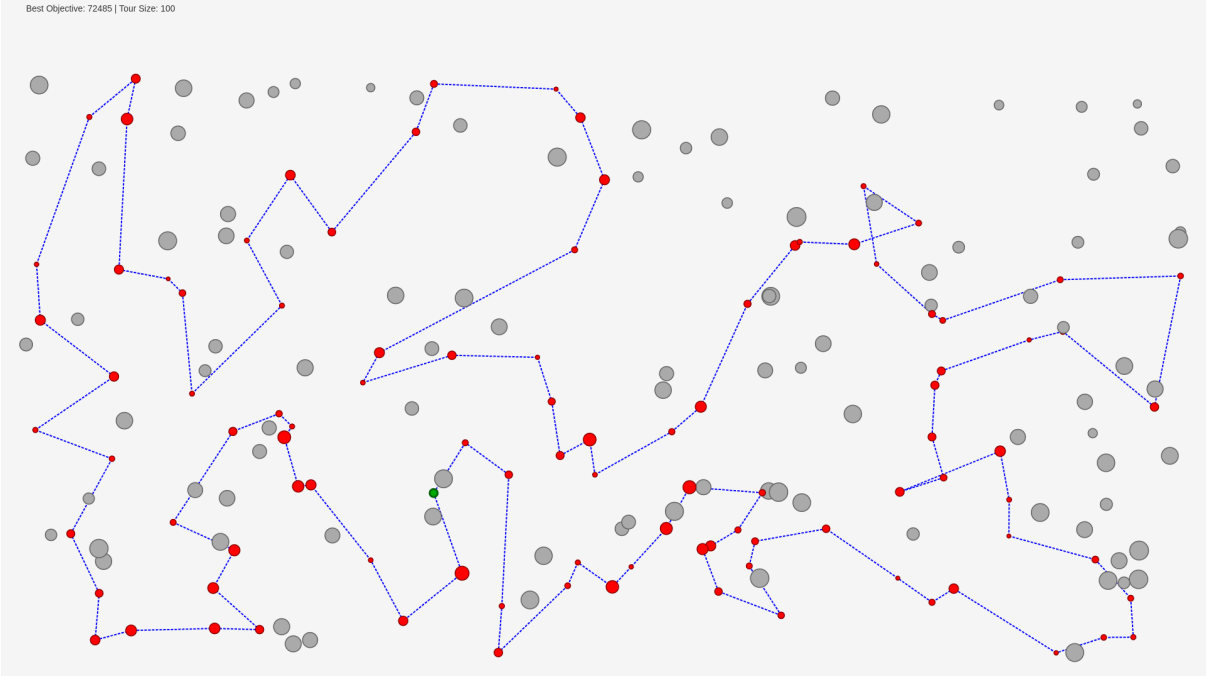
- **Steepest LS 2-opt, rand init**
  - Instance A



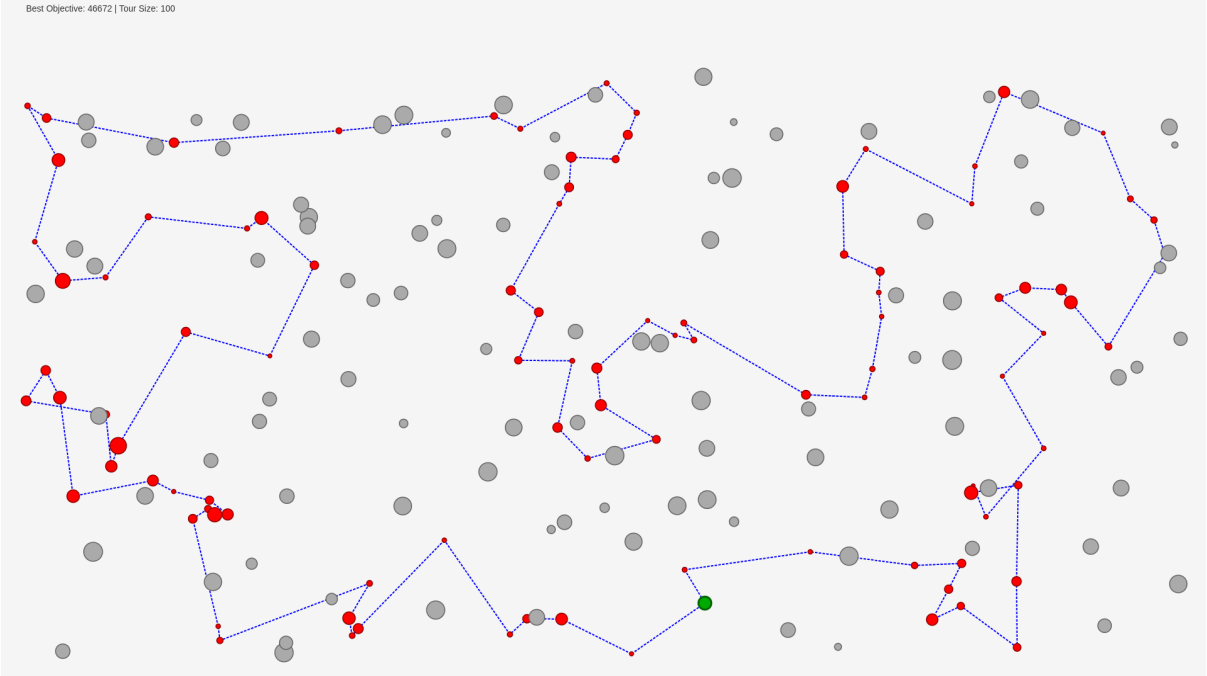
  - Instance B



- **Steepest LS 2-opt, rand init with LM**
    - Instance A



    - Instance B



Best Solutions

The best solutions were checked with the solution checker.

- **Steepest LS 2-opt, rand init**
    - Instance A

116, 5, 42, 43, 35, 184, 177, 54, 48, 160, 34, 181, 146, 22, 159, 193, 41, 115, 46, 68, 139, 69, 18, 108, 140, 93, 117, 0, 143, 183, 89, 23, 137, 176, 80, 79, 122, 63, 94, 124, 167, 148, 9, 62, 102, 49, 144, 14, 138, 3, 178, 106, 52, 55, 57, 129, 92, 78, 145, 179, 185, 40, 119, 165, 90, 81, 196, 31, 113, 175, 171, 16, 25, 44, 120, 2, 74, 152, 97, 1, 101, 75, 86, 26, 100, 121, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65

    - Instance B

169, 132, 70, 3, 15, 145, 13, 195, 168, 139, 11, 182, 138, 104, 8, 144, 33, 160, 29, 0, 109, 35, 143, 159, 106, 124, 128, 62, 18, 55, 34, 152, 183, 140, 199, 4, 149, 28, 20, 60, 148, 47, 94, 179, 99, 130, 95, 185, 86, 166, 176, 113, 114, 137, 127, 89, 103, 163, 187, 153, 81, 77, 111, 82, 21, 141, 91, 61, 36, 177, 5, 78, 175, 45, 80, 190, 193, 31, 73, 54, 117, 1, 27, 38, 102, 63, 135, 122, 100, 40, 107, 133, 90, 131, 121, 125, 51, 147, 6, 188

- **Steepest LS 2-opt, rand init with LM**
    - Instance A

51, 151, 176, 80, 79, 133, 162, 135, 70, 154, 180, 63, 94, 121, 53, 100, 26, 86, 75, 101, 152, 97, 1, 2, 120, 44, 25, 82, 129, 92, 145, 78, 16, 171, 175, 113, 56, 31, 196, 81, 90, 27, 39, 165, 119, 40, 185, 179, 57, 55, 52, 106, 178, 3, 49, 102, 138, 14, 144, 62, 9, 148, 15, 114, 186, 137, 23, 89, 183, 143, 0, 46, 68, 18, 22, 159, 193, 41, 139, 115, 42, 160, 34, 48, 54, 177, 10, 184, 84, 112, 127, 123, 149, 131, 47, 65, 43, 116, 59, 118

    - Instance B

31, 54, 121, 131, 90, 51, 147, 10, 133, 122, 107, 40, 63, 135, 38, 27, 1, 198, 117, 193, 190, 80, 142, 78, 175, 36, 141, 77, 153, 187, 163, 103, 89, 127, 114, 113, 176, 180, 194, 166, 86, 185, 95, 130, 99, 22, 179, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 55, 128, 124, 106, 143, 62, 18, 34, 170, 152, 53, 184, 155, 189, 3, 70, 145, 132, 13, 195, 168, 169, 188, 6, 134, 139, 11, 182, 138, 33, 144, 160, 29, 109, 0, 35, 111, 8, 82, 21, 177, 5, 73

Conclusions

- The LM system reduced the time efficiency drastically, more then the candidate mechanism even.
- The results for instance A improved when compared to candidate LS however they got worse for instance B