Evolutionary Computation
Large Neighborhood Search Report

Authors and Source Code

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

- **Source Code Repository:**

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)

In both cases we use Local Search Steepest with 2-opt move as intra-move.

- **Main LNS loop**

```
function lns(d, nodes, time_limit_ms, useLS):
   # 1. Generate Initial Solution
   start_tour = local search from random solution
   # 2. LNS Loop
   repeat until time_elapsed > time_limit_ms:
      destroyed_tour = destroyLNS(best_solution, d, nodes)
      repaired_tour = repairLNS(destroyed_tour, d, nodes)
      if useLS is true:
         improved_solution = apply local_search to repaired_tour
      else:
         improved_solution = repaired_tour
      obj = compute objective of improved_solution
      if obj < best_obj:
         best_obj = obj
         best_solution = improved_solution
   end repeat
   return best_solution
end function
```

- **Destroy function**

```
function destroyLNS(tour, d, nodes):
    current_k = size of tour
    remove_count = 30% of current_k
    initialize list node_scores
    # 1. Calculate stochastic cost for each node
    for each node i in tour:
        valid_cost = d[prev][i] + d[i][next] + nodes[i].cost
        noisy_cost = valid_cost * random_double(0.8, 1.2) # Randomization
        add (noisy_cost, index_of_i) to node_scores
    end for
    # 2. Sort nodes by cost (highest/worst first)
    sort node_scores descending by noisy_cost
    # 3. Create set of indices to remove
    indices_to_remove = select top 30% remove_count indices from node_scores
    # 4. Rebuild the tour excluding selected nodes
    initialize list destroyed_tour
    for each index j from 0 to current_k - 1:
        if j is not in indices_to_remove:
            add tour[j] to destroyed_tour
    return destroyed_tour
end function
```

- **Repair Function (Greedy cycle regret)**

```
function repairLNS(broken_tour, d, nodes):
    current_tour = broken_tour
    target_k = (total_nodes + 1) / 2
    mark nodes present in current_tour as used
    w1 = 0.5 # Weights for regret
    w2 = 0.5 # Weights for regret
    while size of current_tour < target_k:
        best_candidate = -1
        best_insert_pos = -1
        max_weighted_score = -∞
        # Evaluate every unused node
        for each candidate_node not in used:
            best_cost_increase = +∞
            second_best_cost_increase = +∞
            best_pos_for_cand = -1
            # Check all insertion positions in the current partial tour
            for each position i in current_tour:
                u = current_tour[i]
                v = current_tour[i+1] # (wrapping around)
                cost_increase = d[u][candidate_node] + d[candidate_node][v] - d[u][v] +
nodes[candidate_node].cost
                if cost_increase < best_cost_increase:
                    second_best_cost_increase = best_cost_increase
                    best_cost_increase = cost_increase
                    best_pos_for_cand = i
                else if cost_increase < second_best_cost_increase:
```

```
            second_best_cost_increase = cost_increase
        # Calculate Weighted Regret Score
        regret = second_best_cost_increase - best_cost_increase
        score = (w1 * regret) - (w2 * best_cost_increase)
        if score > max_weighted_score:
            max_weighted_score = score
            best_candidate = candidate_node
            best_insert_pos = best_pos_for_cand
      end for
      # Insert the best candidate found
      insert best_candidate into current_tour at position best_insert_pos + 1
      mark best_candidate as used
   end while
   return current_tour
end function
```

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)
Multiple Start Local Search	**71291.8 (70550 - 71909)**	**45738.9 (45005 - 46259)**
Iterated Local Search	**69220.9 (69095 - 69653)**	**43606.8 (43446 - 44125)**
LNS without LS after destroy/repair	**70791.5 (70227 - 71575)**	**45361.6 (43830 - 46485)**
LNS with LS after destroy/repair	**70393.9 (69437 - 71007)**	**44687.9 (43652 - 46114)**

Number of iterations of main loop

INSTANCE A	
Without LS after destroy/repair	With LS after destroy/repair
2064.2	1774.4
INSTANCE B	
Without LS after destroy/repair	With LS after destroy/repair
2084.8	1664.75

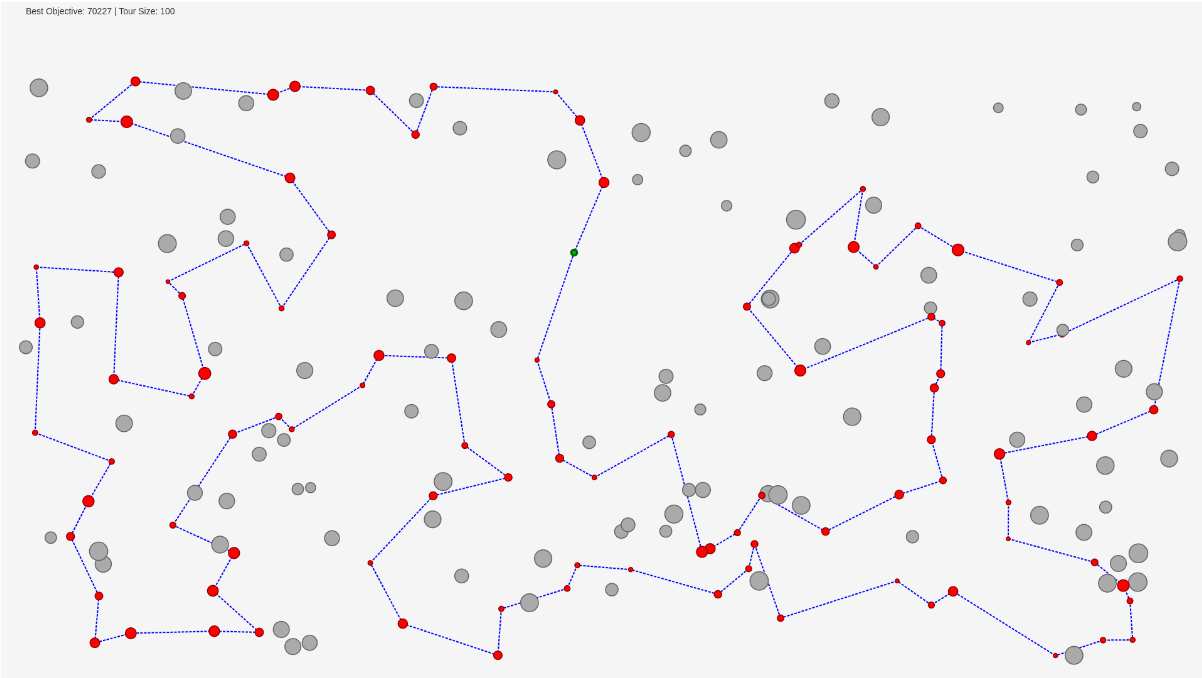
Table of Runtimes

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
Multiple Start Local Search	**1.688**	**1.422**
Iterated Local Search	**1.689**	**1.424**
LNS without LS after destroy/repair	**1.43935**	**1.44235**
LNS with LS after destroy/repair	**1.4394**	**1.4424**

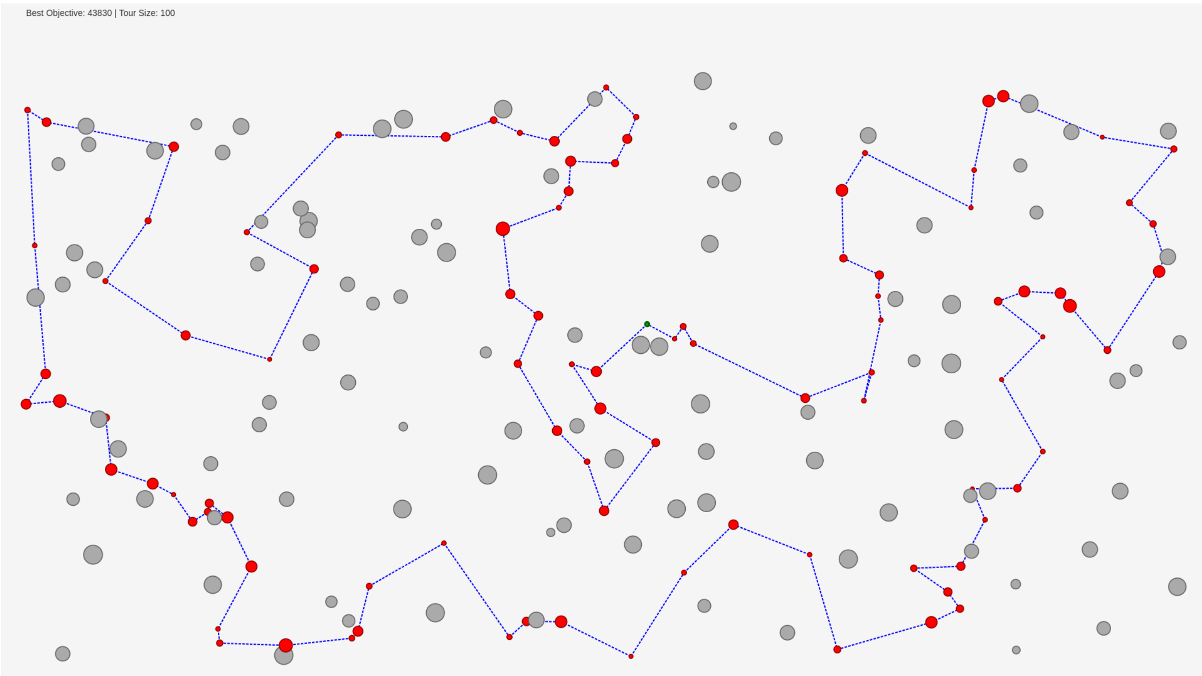
Visual Comparisons (Visual Comparision)

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.

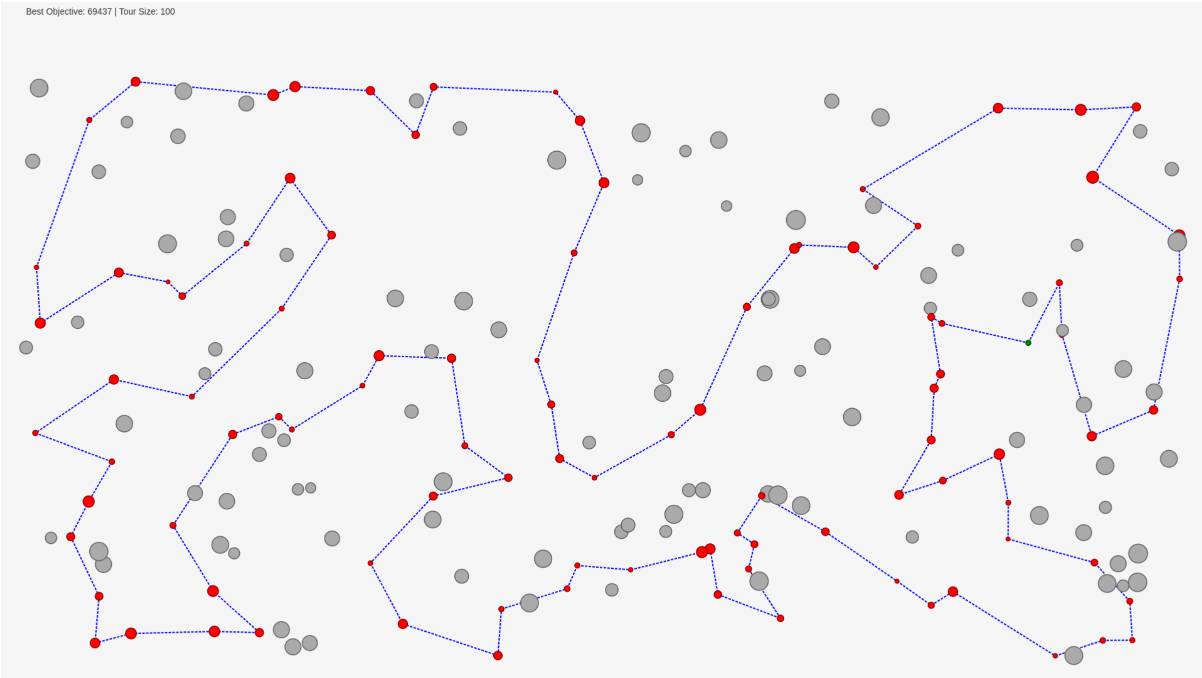
- **LNS without LS after destroy/repair**
  - Instance A



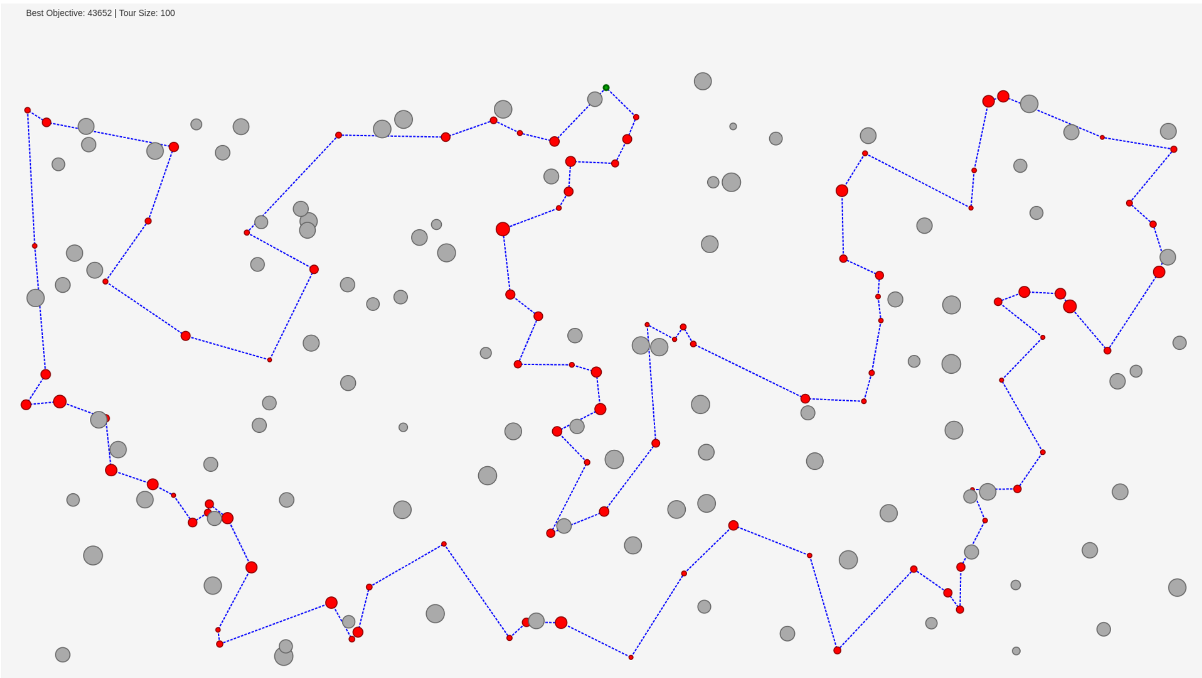
  - Instance B



- **LNS with LS after destroy/repair**
    - Instance A



    - Instance B



Best Solutions

The best solutions were checked with the solution checker.

- **LNS without LS after destroy/repair**
  - Instance A

137, 176, 80, 79, 63, 94, 100, 26, 97, 152, 2, 129, 92, 57, 55, 52, 106, 178, 167, 148, 9, 62, 144, 102, 49, 14, 138, 165, 185, 40, 90, 81, 196, 179, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 75, 1, 101, 86, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 146, 22, 159, 181, 42, 5, 41, 193, 139, 115, 46, 68, 69, 18, 108, 140, 93, 117, 0, 143, 183, 89, 23

  - Instance B

29, 160, 33, 144, 111, 82, 8, 104, 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, 165, 127, 89, 163, 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, 106, 124, 143, 35, 109, 0

- **LNS with LS after destroy/repair**
  - Instance A

185, 165, 40, 196, 81, 90, 27, 95, 164, 7, 21, 144, 14, 49, 102, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 89, 183, 143, 0, 117, 93, 140, 108, 18, 22, 146, 159, 193, 41, 139, 68, 46, 115, 42, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 184, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25, 16, 171, 175, 113, 31, 78, 145, 179, 92, 129, 57, 55, 52, 178, 106

  - Instance B

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, 175, 78, 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, 21, 8, 104, 144, 160, 33, 138, 11, 139, 43, 168, 195, 13, 145, 15, 3

Conclusions

- As expected the LNS with LS achieves better results then LNS without LS.
- Both of the new methods did not outperform ILS, however perhaps there is room for improvement by tuning the noise distribution or the % of nodes to be removed.
- The LNS methods did however run faster then the ILS.