Evolutionary Computation
MSLS & ILS Report

Authors and Source Code

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

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

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.

- **Multiple Start Local Search**

```
function MSLS:
    initialize global_best_obj = +∞
    initialize list msls_best_objectives
    repeat N_EXPERIMENTS times:
        best_obj_this_run = +∞
        repeat ITERATIONS times:
            start_tour = generate random solution
            final_tour = apply local search to start_tour
            obj = compute objective of final_tour
            if obj < best_obj_this_run:
                best_obj_this_run = obj
                best_tour_this_run = final_tour
        end repeat
        record best_obj_this_run in msls_best_objectives
        if best_obj_this_run < global_best_obj:
            global_best_obj = best_obj_this_run
            global_best_tour = best_tour_this_run
    end repeat
    return  global_best_tour
end function
```

- **Iterated Local Search**

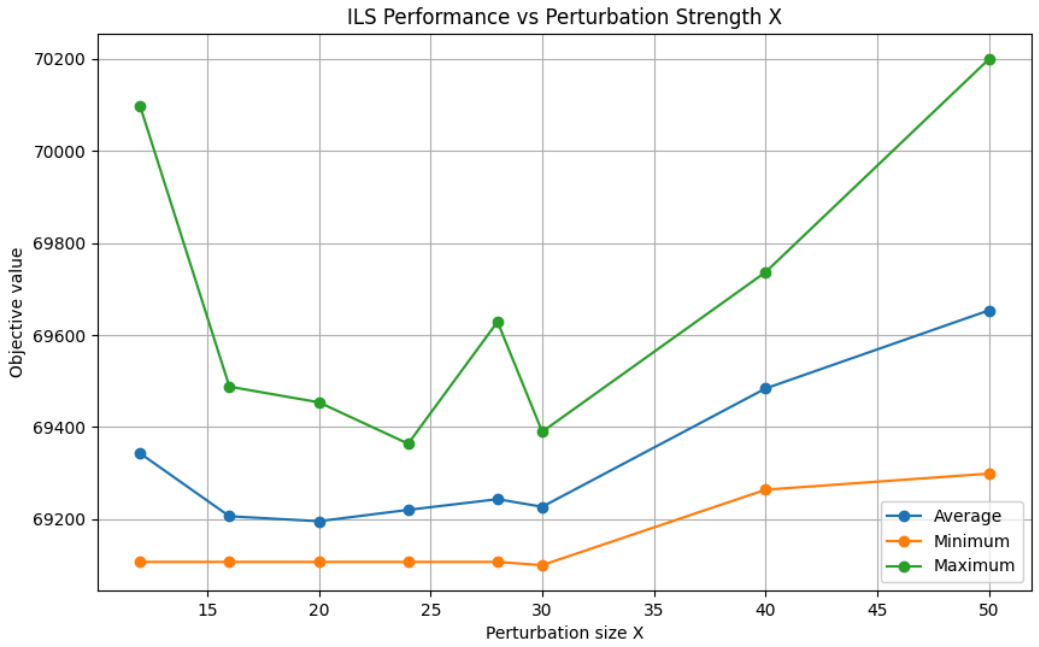
```
function PERTURB(tour, X): // X is the number of nodes in tour to be swapped (20 in our case)
    new_tour ← copy of tour
    num_swaps ← X / 2
    repeat num_swaps times:
        choose two random positions i and j
        if i ≠ j:
            swap new_tour[i], new_tour[j]
    end repeat
    return new_tour
end function
```

```
function ILS(distance_matrix, nodes, time_limit, X):
    repeat N_EXPERIMENTS times:
        start measuring time for this run
        current ← random solution
        current ← local search applied to current
        best ← current
        ls_runs ← 1
        while time for this run < time_limit:
            perturbed ← PERTURB(best, X)
            improved ← local search applied to perturbed
            ls_runs++
            if improved is better than best:
                best ← improved
            end if
        end while
```

```
        record best objective for this run
        record ls_runs
        update global best if necessary
    end repeat
    compute statistics over all runs
    export global best solution
end function
```

The perturb function randomly swaps $X/2$ pairs of nodes in the current solution, affecting a total of X node positions. This introduces a controlled amount of disruption that breaks the current local optimum without destroying all solution structure. It enables ILS to escape local minima and explore new regions before applying local search again.

Experimentally we established that the best X will be 20.



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)**

Instance A

```
ILS run 1: best obj=69213 | LS calls=785
ILS run 2: best obj=69095 | LS calls=675
ILS run 3: best obj=69162 | LS calls=680
ILS run 4: best obj=69154 | LS calls=837
ILS run 5: best obj=69165 | LS calls=1032
ILS run 6: best obj=69414 | LS calls=933
ILS run 7: best obj=69137 | LS calls=891
ILS run 8: best obj=69283 | LS calls=983
ILS run 9: best obj=69095 | LS calls=928
ILS run 10: best obj=69107 | LS calls=898
ILS run 11: best obj=69140 | LS calls=888
ILS run 12: best obj=69107 | LS calls=928
ILS run 13: best obj=69235 | LS calls=1080
ILS run 14: best obj=69223 | LS calls=1077
ILS run 15: best obj=69334 | LS calls=1007
ILS run 16: best obj=69200 | LS calls=973
ILS run 17: best obj=69107 | LS calls=944
ILS run 18: best obj=69202 | LS calls=1039
ILS run 19: best obj=69653 | LS calls=1023
ILS run 20: best obj=69392 | LS calls=947
```

Instance B

```
ILS run 1: best obj=43448 | LS calls=1071
ILS run 2: best obj=43605 | LS calls=1097
ILS run 3: best obj=43462 | LS calls=1091
ILS run 4: best obj=43487 | LS calls=1078
ILS run 5: best obj=43456 | LS calls=953
ILS run 6: best obj=43602 | LS calls=963
ILS run 7: best obj=43981 | LS calls=967
ILS run 8: best obj=43509 | LS calls=969
ILS run 9: best obj=43446 | LS calls=977
ILS run 10: best obj=43524 | LS calls=981
ILS run 11: best obj=43944 | LS calls=974
ILS run 12: best obj=43446 | LS calls=954
ILS run 13: best obj=44125 | LS calls=982
ILS run 14: best obj=43556 | LS calls=975
ILS run 15: best obj=43503 | LS calls=982
ILS run 16: best obj=43567 | LS calls=967
ILS run 17: best obj=43496 | LS calls=980
ILS run 18: best obj=43510 | LS calls=985
ILS run 19: best obj=43467 | LS calls=981
ILS run 20: best obj=44002 | LS calls=980
```

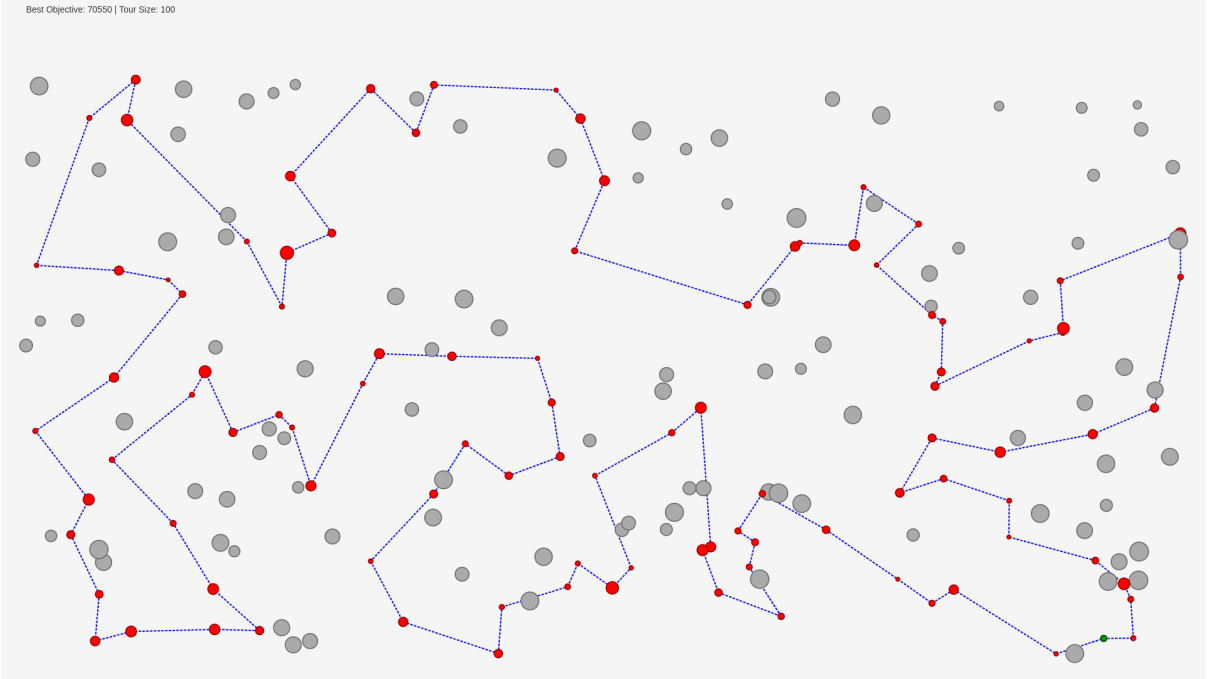
Table of Runtimes

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	**33.76 s**	**28.446**
Iterated Local Search	**33.782 s**	**28.475**

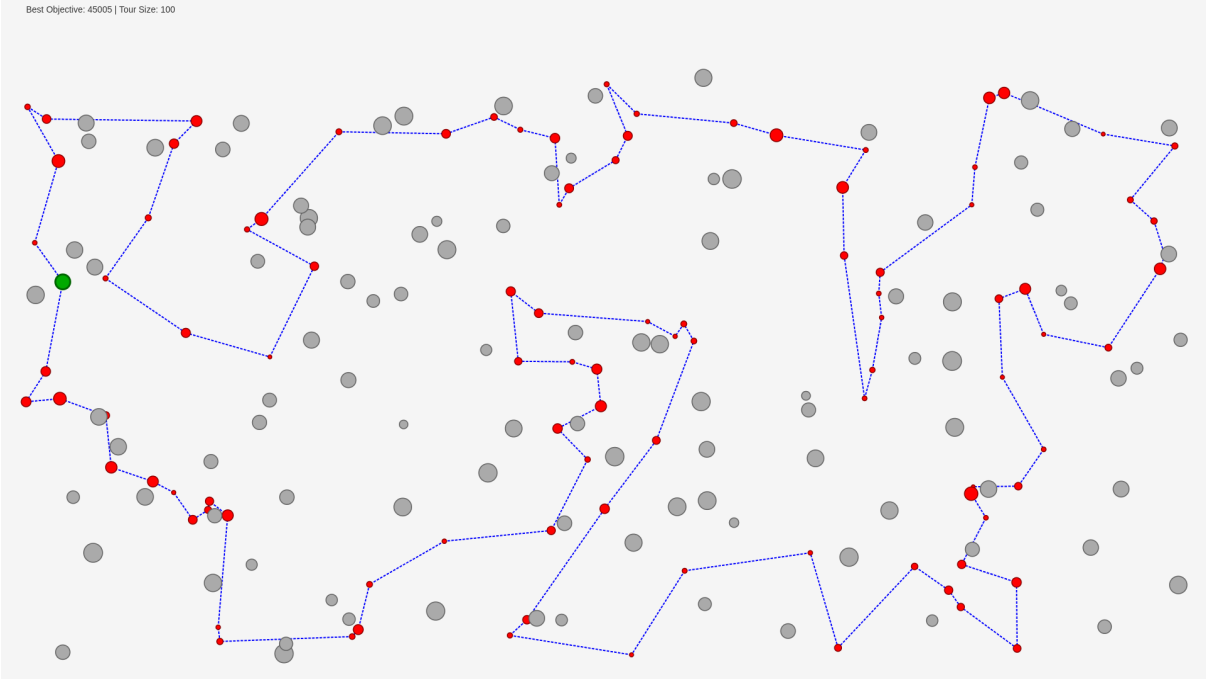
Visual Comparisons (Visual Comparision)

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.

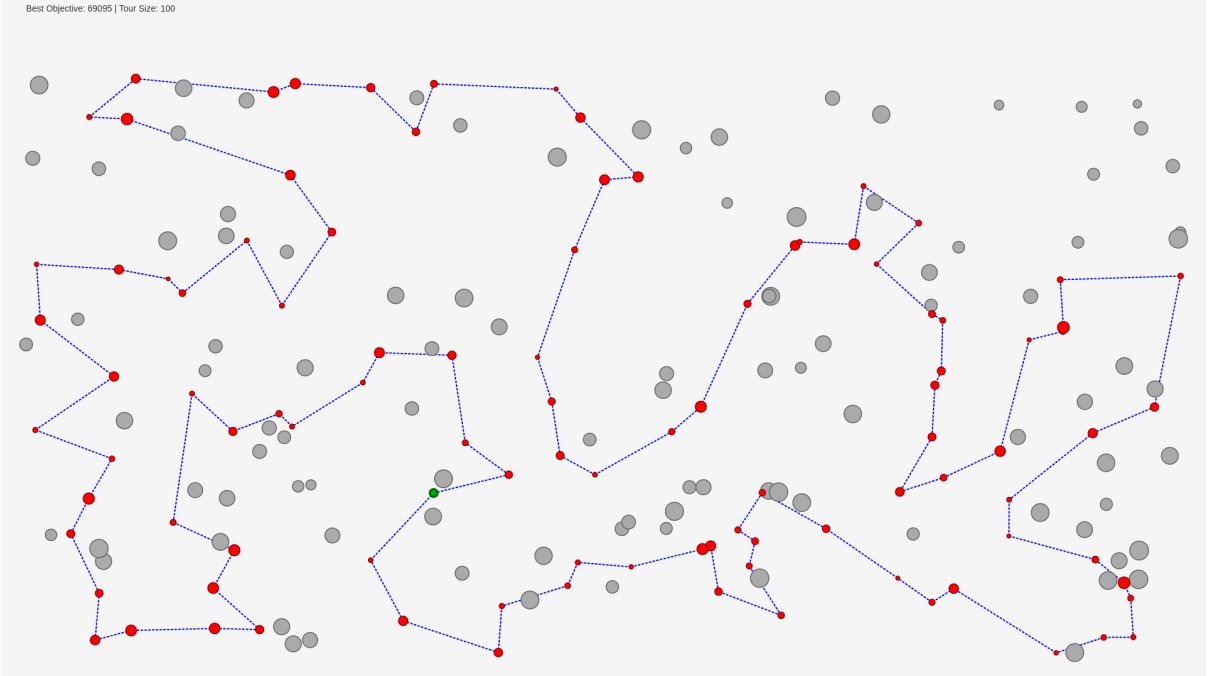
- **Multiple Start Local Search**
    - Instance A



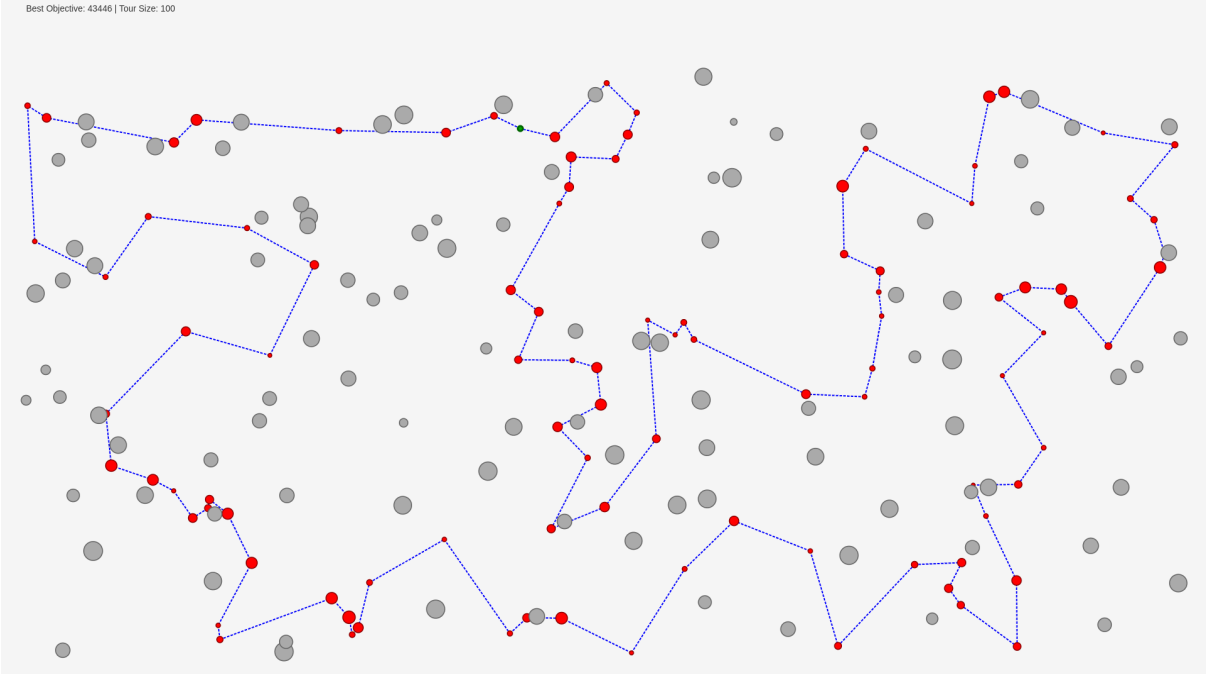
    - Instance B



- **Iterated Local Search**
    - Instance A



    - Instance B



Best Solutions

The best solutions were checked with the solution checker.

- **Multiple Start Local Search**
    - Instance A

171, 175, 113, 56, 31, 78, 145, 92, 129, 57, 179, 196, 81, 90, 27, 165, 119, 40, 185, 55, 52, 106, 178, 49, 14, 144, 102, 62, 9, 148, 137, 23, 89, 183, 143, 0, 117, 68, 46, 198, 115, 139, 69, 108, 18, 22, 159, 193, 41, 181, 34, 48, 54, 177, 10, 190, 4, 112, 84, 184, 160, 42, 5, 43, 116, 65, 149, 59, 118, 51, 176, 80, 79, 133, 151, 162, 123, 127, 70, 135, 154, 180, 158, 53, 63, 94, 124, 26, 100, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25, 16

    - Instance B

102, 63, 100, 40, 107, 10, 133, 122, 135, 131, 121, 51, 90, 191, 147, 6, 188, 169, 132, 168, 195, 145, 15, 70, 3, 155, 184, 152, 170, 34, 106, 124, 62, 18, 55, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 179, 185, 130, 95, 86, 166, 194, 176, 180, 113, 103, 114, 137, 127, 89, 163, 187, 153, 77, 141, 36, 61, 82, 111, 35, 109, 0, 29, 11, 139, 138, 33, 160, 144, 104, 8, 21, 177, 5, 78, 175, 80, 190, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 38

- **Iterated Local Search**
    - Instance A

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, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 57, 129, 92, 179, 185, 40, 119, 165, 90, 81, 196, 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

    - Instance B

169, 188, 6, 147, 10, 133, 107, 40, 63, 135, 122, 90, 51, 121, 131, 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, 103, 89, 127, 137, 114, 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, 168, 195, 13, 145, 15, 3, 70, 132

Conclusions

- **Superior Performance of ILS:** The Iterated Local Search algorithm consistently produced the best overall objective values for both instances outperforming all other methods.
- **ILS Effectiveness:** The ILS method, utilizing the specialized `PERTURB` function (randomly swapping 10 pairs of nodes) to escape local optima, proved more effective in finding superior solutions compared to the simpler multiple random restarts of MSLS.