Evolutionary Computation
Local Search Report

Authors and Source Code

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

- **Source Code Repository:** https://github.com/MZmuda-Trzebiatowski/Evolutionary-Computiation-Local-Search

---

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**
  - Input:
    - tour: list of node indices representing the current solution
    - d: distance matrix between nodes
    - nodes: list of nodes, each with an associated cost
    - use_swap_intra: boolean flag indicating whether to use swap or 2-opt for intra-route optimization
  - Output:
    - tour: improved route after applying local search

```
Let n ← size(d) and k ← size(tour).
Initialize a boolean array is_selected of size n ← false.
For each node v in tour, set is_selected[v] ← true.
Repeat
    a. Initialize best_move.delta ← 0.
    b. Inter-route improvement (V–E exchange):
        For each position i in tour do
            For each unselected node j do
                Compute delta ← delta_V_E_exchange(tour, i, j, d, nodes).
                If delta < best_move.delta then
                    Update best_move ← {type = 0, i, j, delta}.
    c. Intra-route improvement (Swap or 2-opt):
        For each pair of indices (i, j) with i < j do
            If use_swap_intra = true, set delta ← delta_swap(tour, i, j, d) and type ← 1.
            Else, set delta ← delta_2opt(tour, i, j, d) and type ← 2.
            If delta < best_move.delta then
                Update best_move ← {type, i, j, delta}.
    d. If best_move.delta ≥ 0, then break.
    e. Apply the best move found:
        - If best_move.type = 0 (V–E exchange):
            Remove node at position i, insert node j using apply_V_E_exchange.
            Update selection flags in is_selected.
        - If best_move.type = 1 (Swap): apply apply_swap(tour, i, j).
        - If best_move.type = 2 (2-opt): apply apply_2opt(tour, i, j).
Until no improving move exists (best_move.delta ≥ 0).
Return the improved tour.
```

- **Local Search Greedy**
  - Input:
    - tour: list of node indices representing the current solution
    - d: distance matrix between nodes
    - nodes: list of nodes, each with an associated cost

        - use_swap_intra: boolean flag indicating whether to use swap or 2-opt for intra-route optimization
    - Output:
        - tour: improved route after applying local search

Let `n ← size(d)` and `k ← size(tour)`.
Initialize a boolean array `is_selected` of size `n` ← false.
For each node `v` in `tour`, set `is_selected[v] ← true`.
Repeat
a. Set `improved_in_iteration ← false`.
b. Create a list `tour_indices ← [0, 1, …, k-1]`.
c. Create a list `unselected_nodes` containing all nodes not in the current `tour`.
d. Define a list `neighborhood_order ← [0, 1]`, representing move types (*0*: Inter-route, *1*: Intra-route).
e. Randomly shuffle `neighborhood_order`.
f. **For each** `move_type_code` in `neighborhood_order` **do**
If `move_type_code = 0` (*Inter-route search*):
- Randomly shuffle `tour_indices` and `unselected_nodes`.
- **For each** index `i` in `tour_indices` **do**
For each node `u` in `unselected_nodes` **do**
Compute `delta ← delta_V_E_exchange(tour, i, u, d, nodes)`.
If `delta < 0` (*improving move found*):
Apply `apply_V_E_exchange(tour, i, u)`.
Update selection flags:
`is_selected[tour[i]] ← false`, `is_selected[u] ← true`.
Set `improved_in_iteration ← true`.
Go to step (j).
Else if `move_type_code = 1` (*Intra-route search*):
- Randomly shuffle `tour_indices`.
- **For each** index `i` in `tour_indices` **do**
Construct `j_indices ← [i+1, i+2, …, k-1]` and shuffle it.
For each index `j` in `j_indices` **do**
If `use_swap_intra = true`, compute `delta ← delta_swap(tour, i, j, d)` and set `move_type ← 1`.
Else, compute `delta ← delta_2opt(tour, i, j, d)` and set `move_type ← 2`.
If `delta < 0` (*improving move found*):
If `move_type = 1`, apply `apply_swap(tour, i, j)`.
If `move_type = 2`, apply `apply_2opt(tour, i, j)`.
Set `improved_in_iteration ← true`.
Go to step (j).
j. **If** `improved_in_iteration = false`, **then break** the loop.
Until no improving move is found in an iteration.
Return the improved `tour`.

Delta Calculation Functions

- `delta_V_E_exchange(tour, i, u, d, nodes)`
  - Calculates the total change in objective value (distance and node cost) when a selected node in the tour is replaced by an unselected node.
- `delta_swap(tour, i, j, d)`
  - Computes the variation in total travel distance resulting from swapping two nodes within the same tour.
- `delta_2opt(tour, i, j, d)`
  - Determines the difference in travel distance obtained by performing a 2-opt move, which reverses a segment of the tour between two specified edges.

Move Application Functions

- `apply_V_E_exchange(tour, i, u)`
  - Executes a vertex–element exchange by substituting the node at position v_idx with the unselected node e_idx in the tour.
- `apply_swap(tour, i, j)`
  - Applies a swap operation that exchanges the positions of two nodes within the tour sequence.
- `apply_2opt(tour, i, j)`
  - Implements a 2-opt operation by reversing the order of nodes between two selected edges, effectively reconfiguring a segment of the tour.

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)

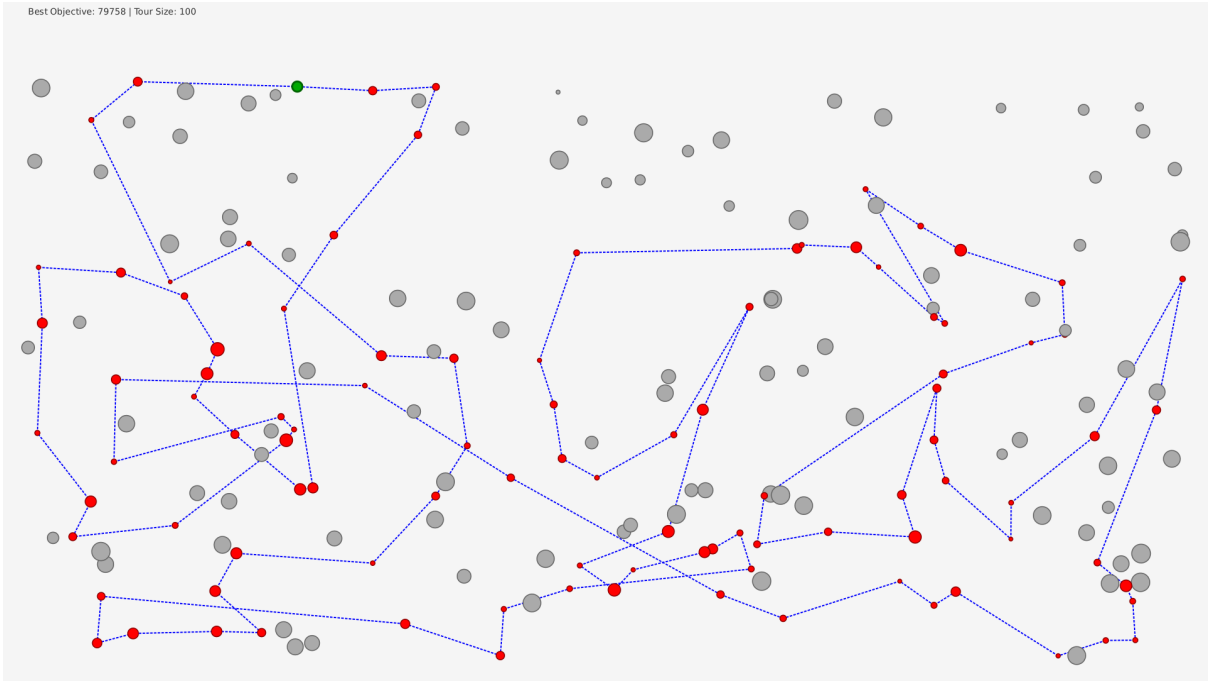
Table of Runtimes

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, 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

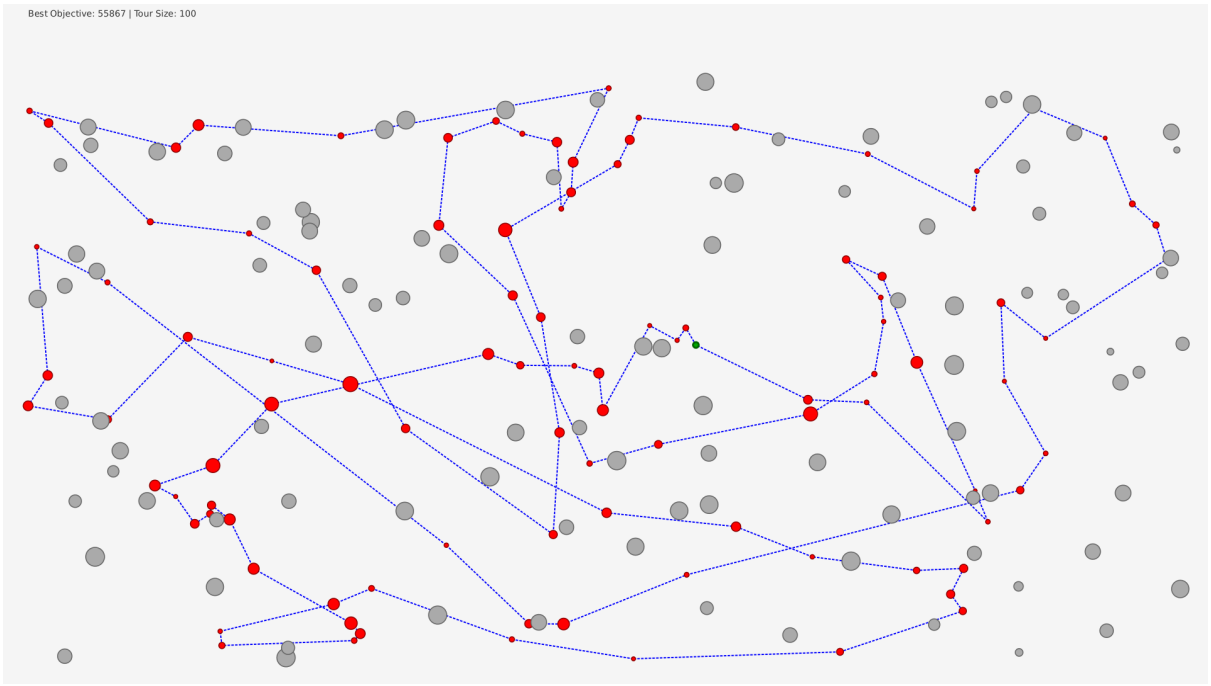
Visual Comparisons (Visual Comparision) - Instance A

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.

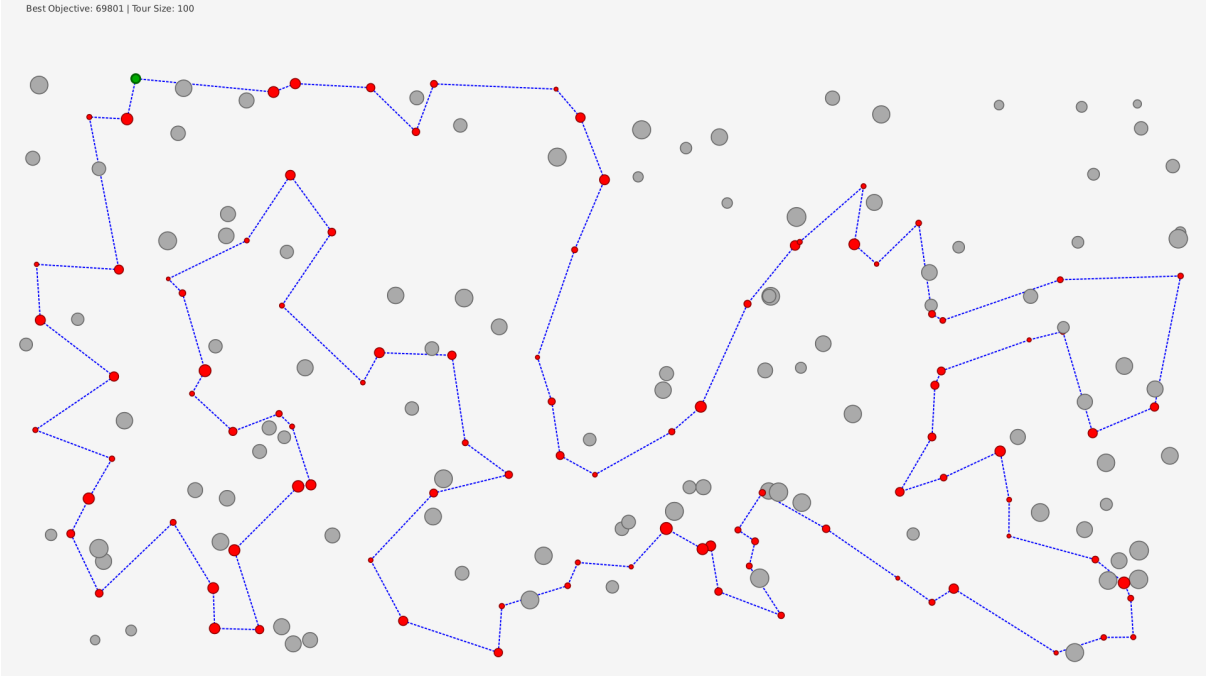
- **Steepest LS swap, rand init**
    - Instance A



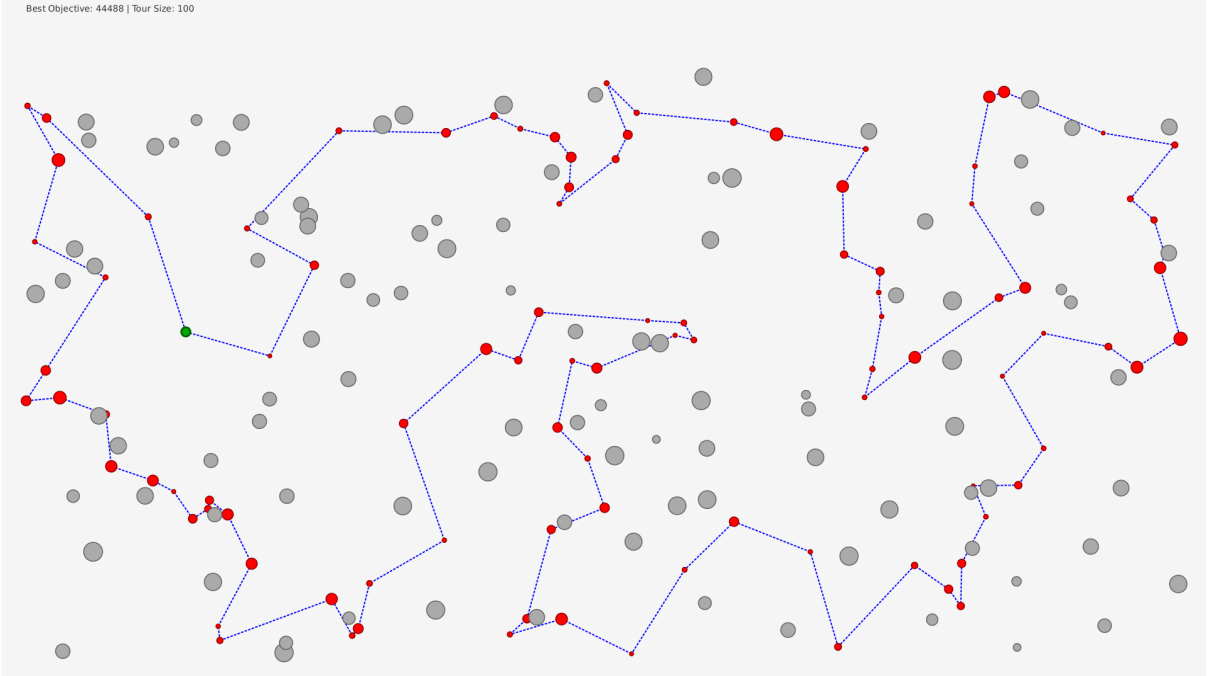
    - Instance B



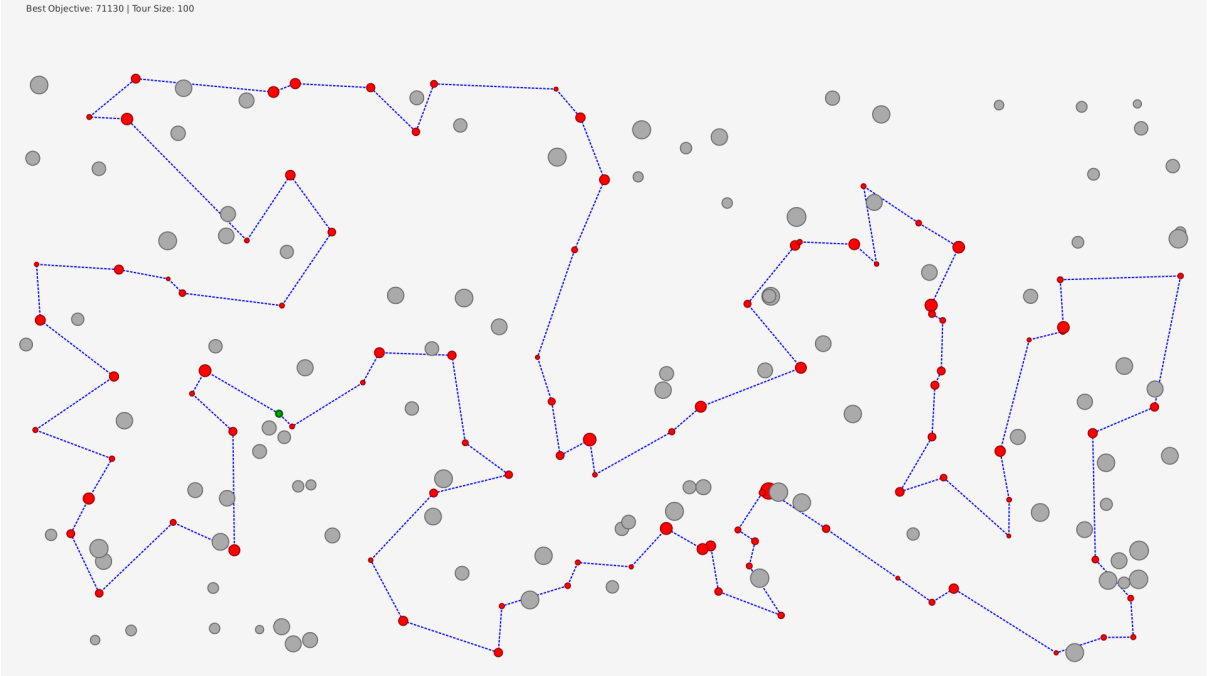
- **Steepest LS swap, greedy init**
  - Instance A



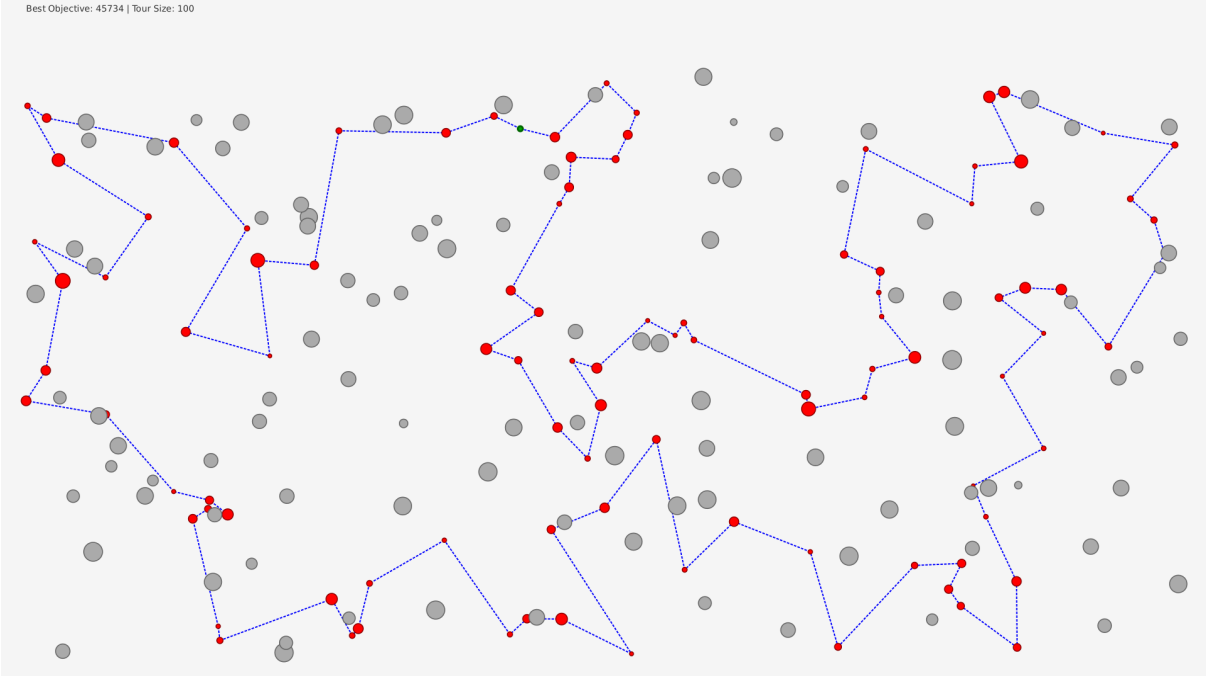
  - Instance B



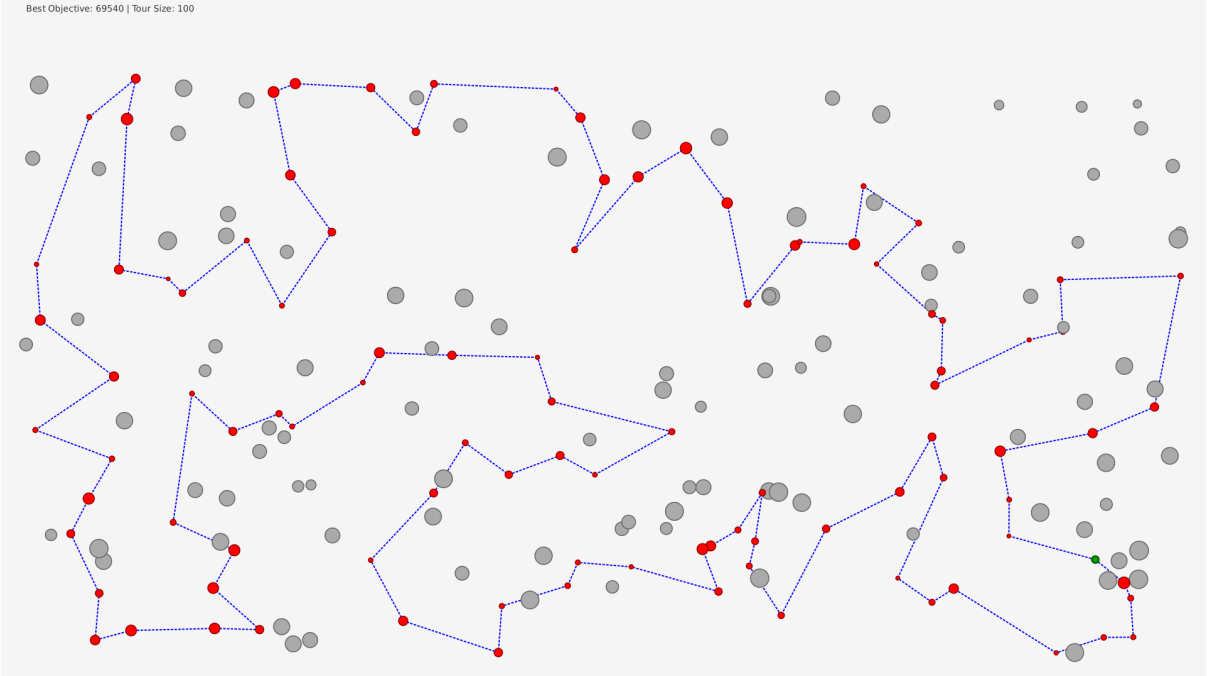
- **Steepest LS 2-opt, rand init**
  - Instance A



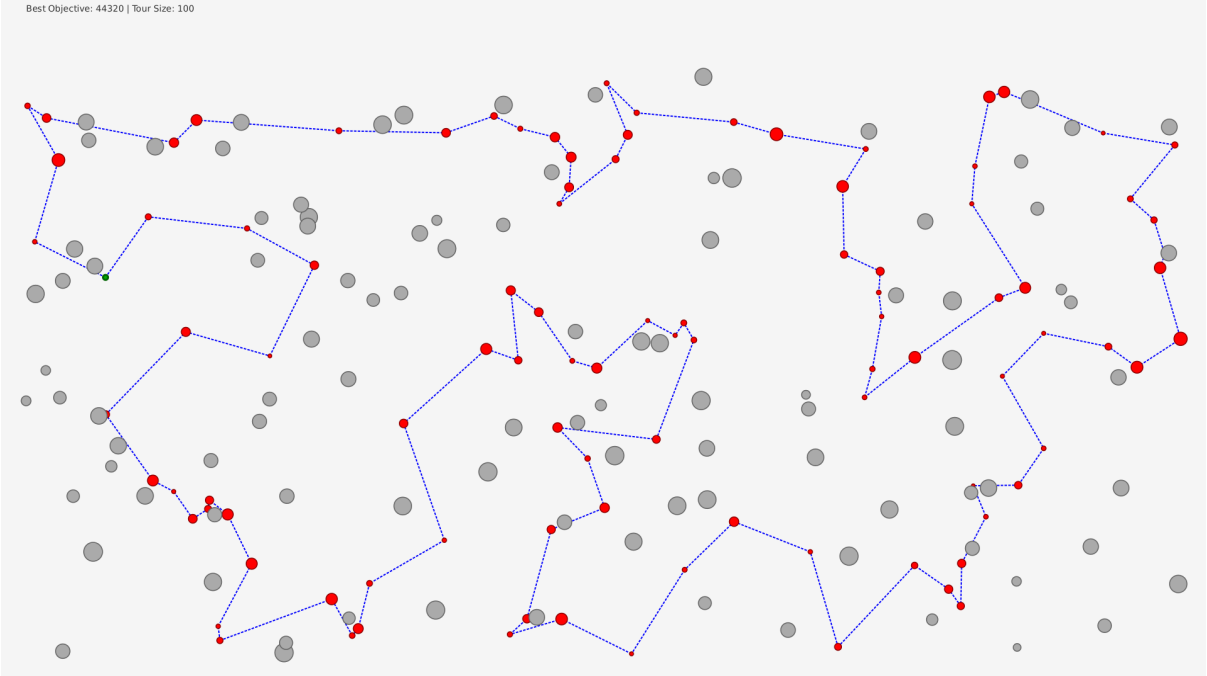
  - Instance B



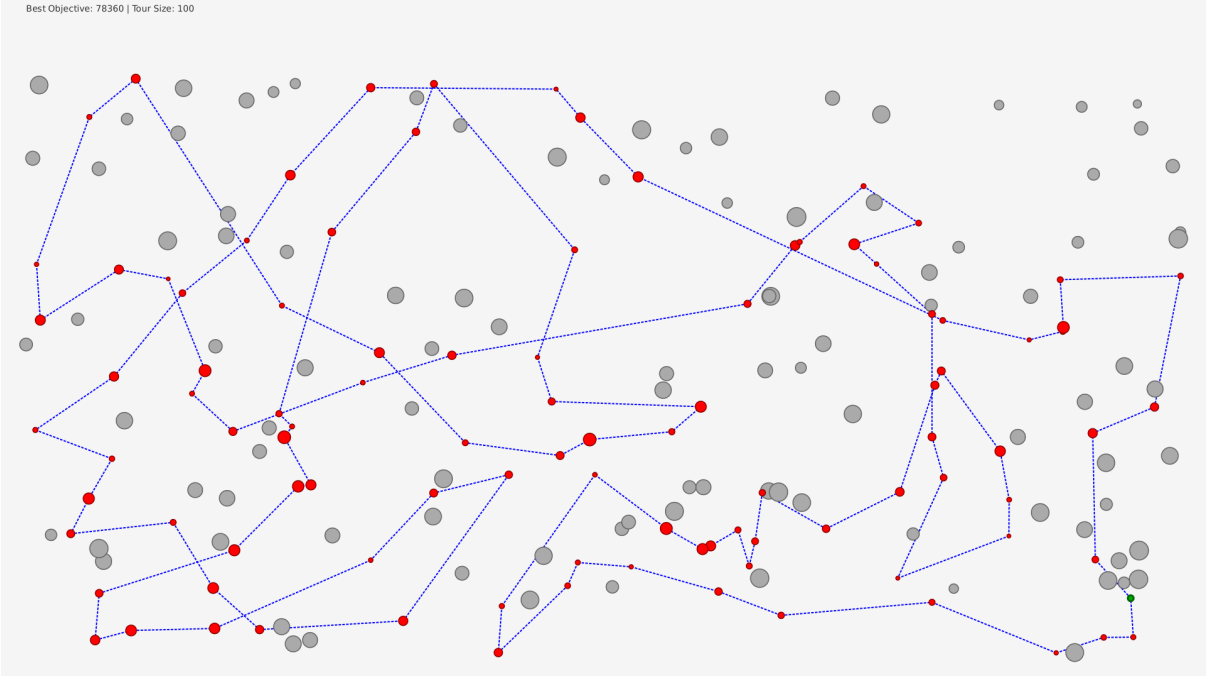
- **Steepest LS 2-opt, greedy init**
    - Instance A



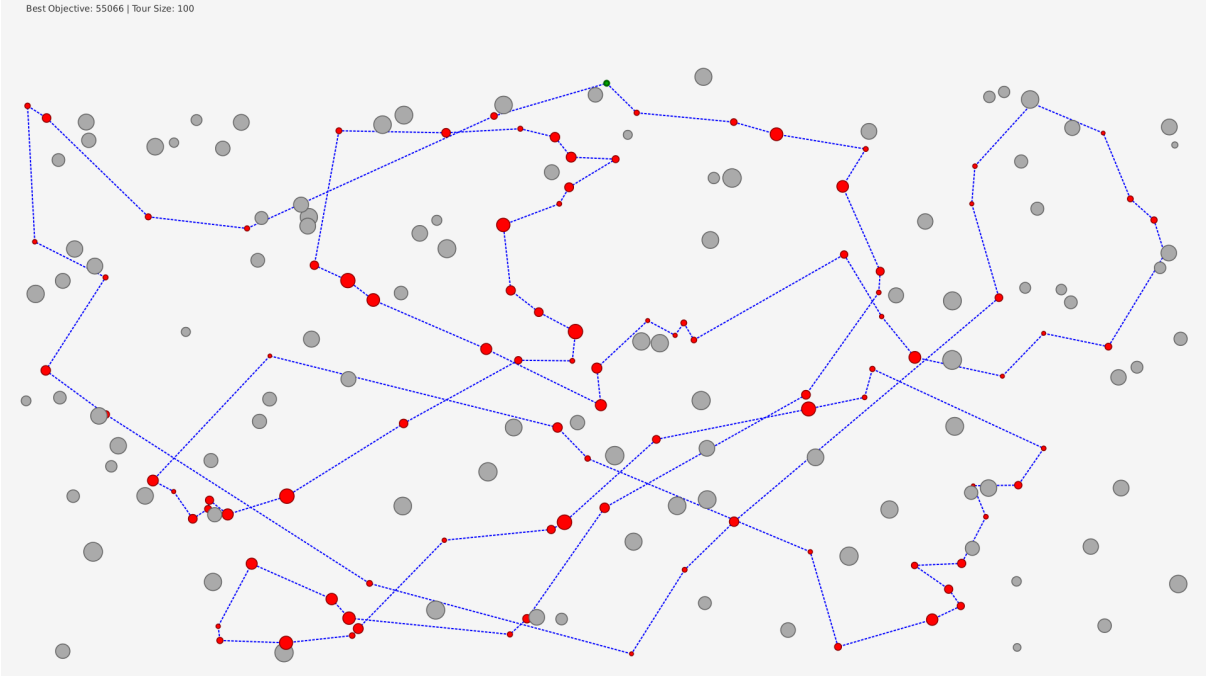
    - Instance B



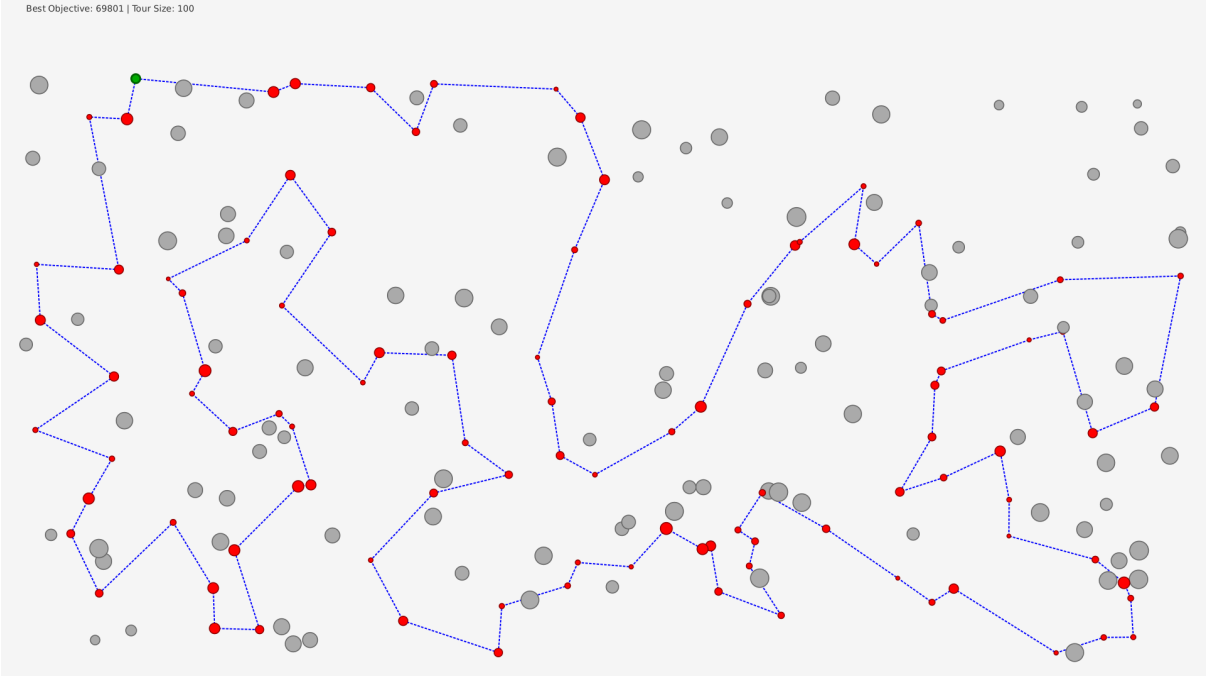
- **Greedy LS swap, rand init**
    - Instance A



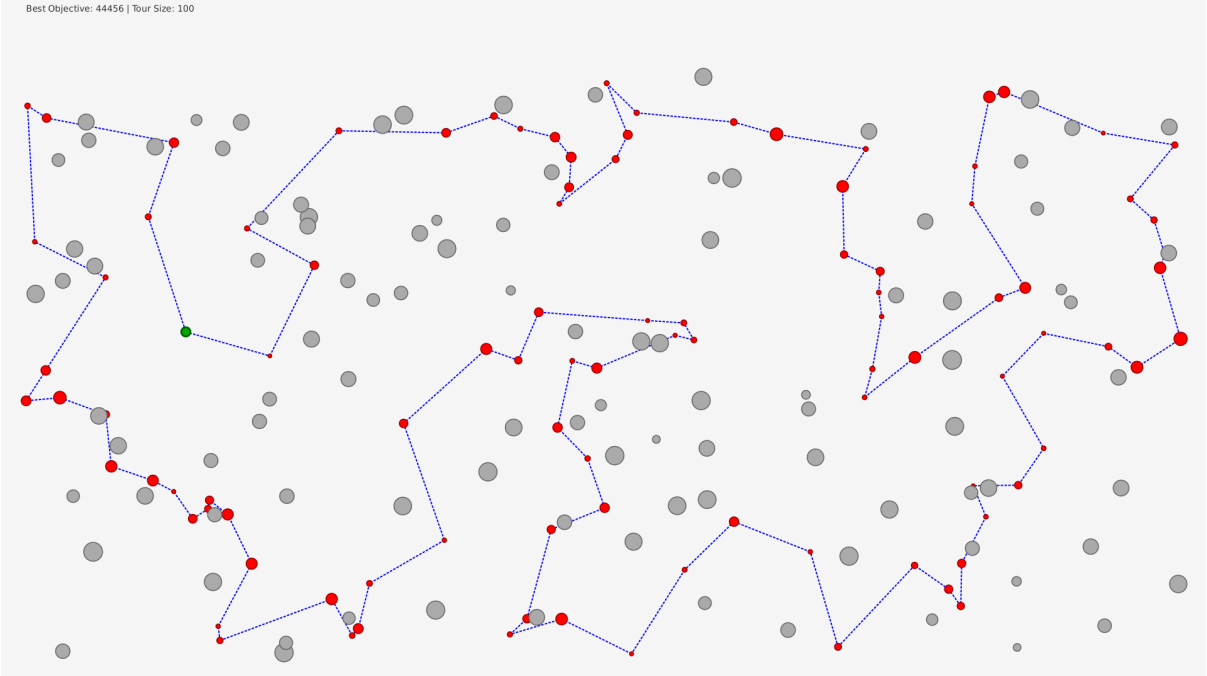
    - Instance B



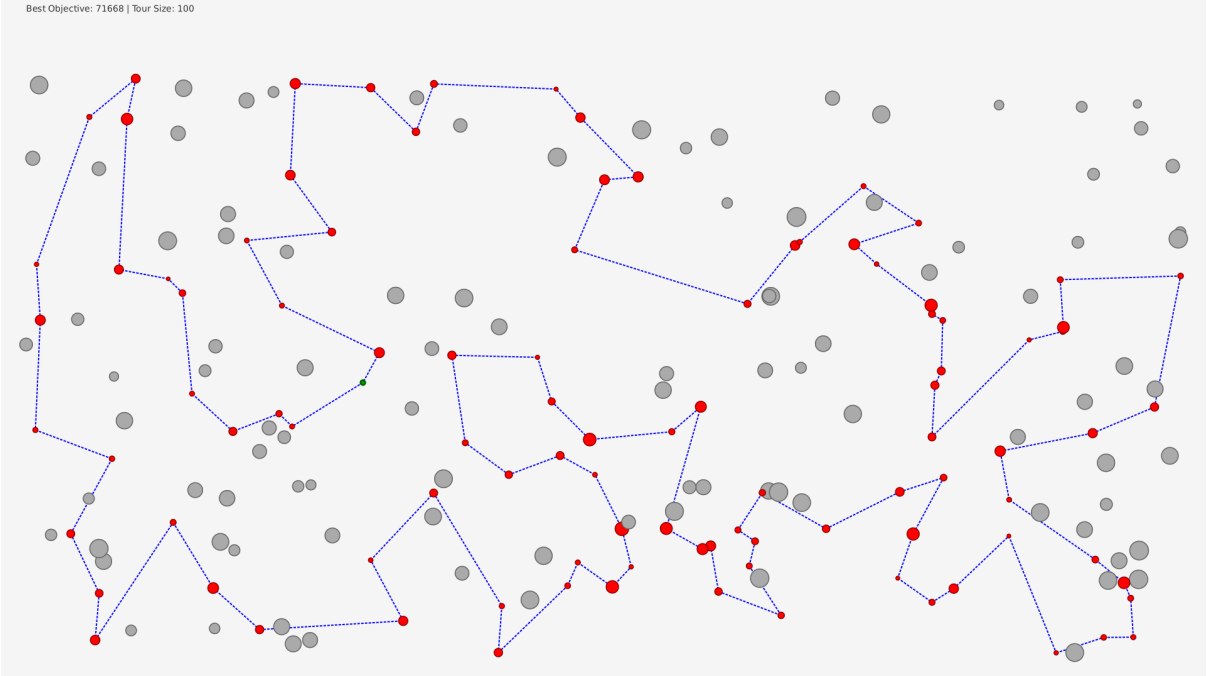
- **Greedy LS swap, greedy init**
    - Instance A



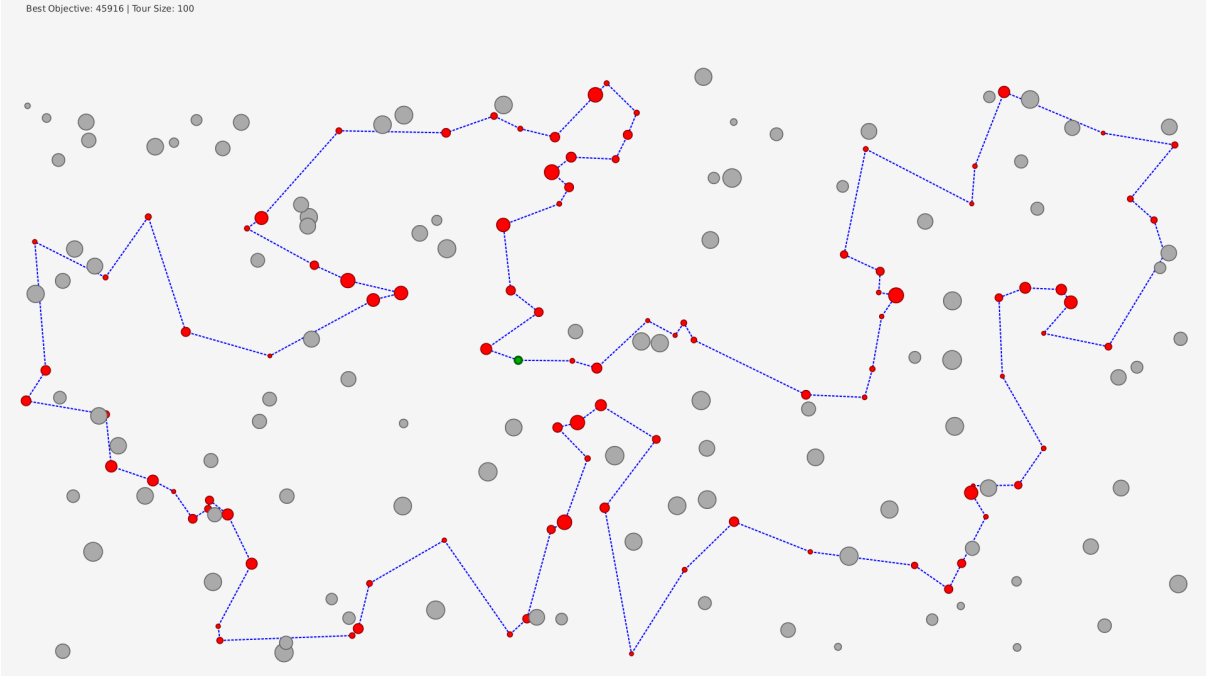
    - Instance B



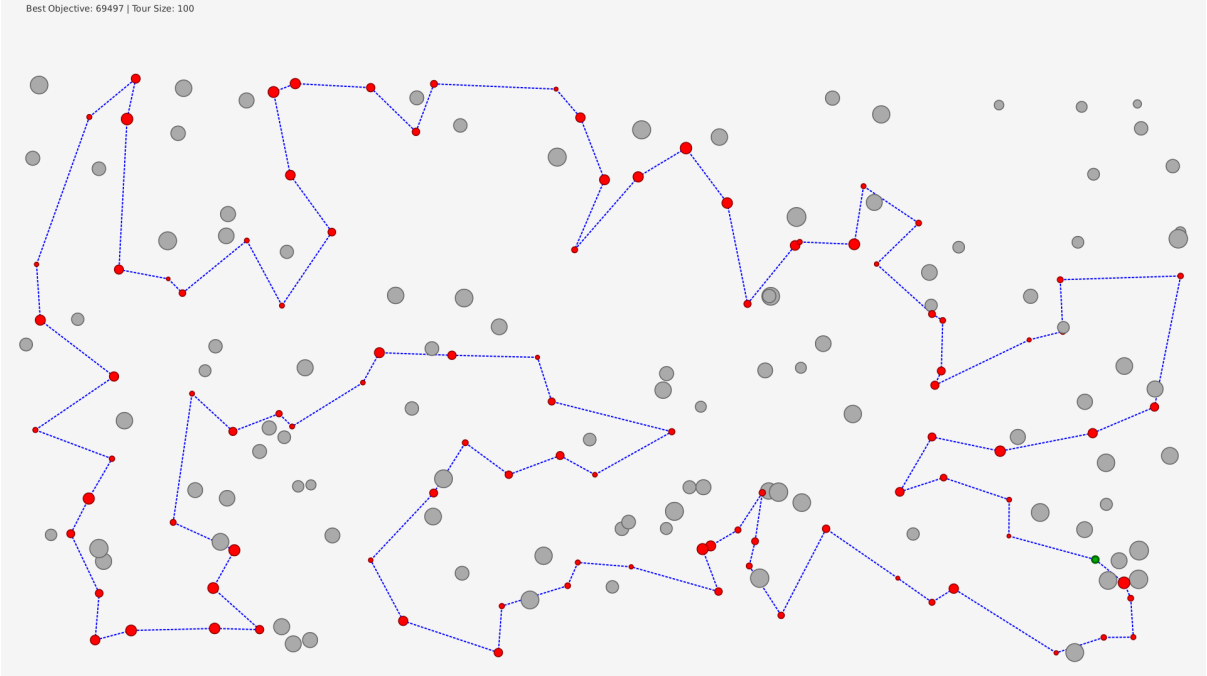
- **Greedy LS 2-opt, rand init**
    - Instance A



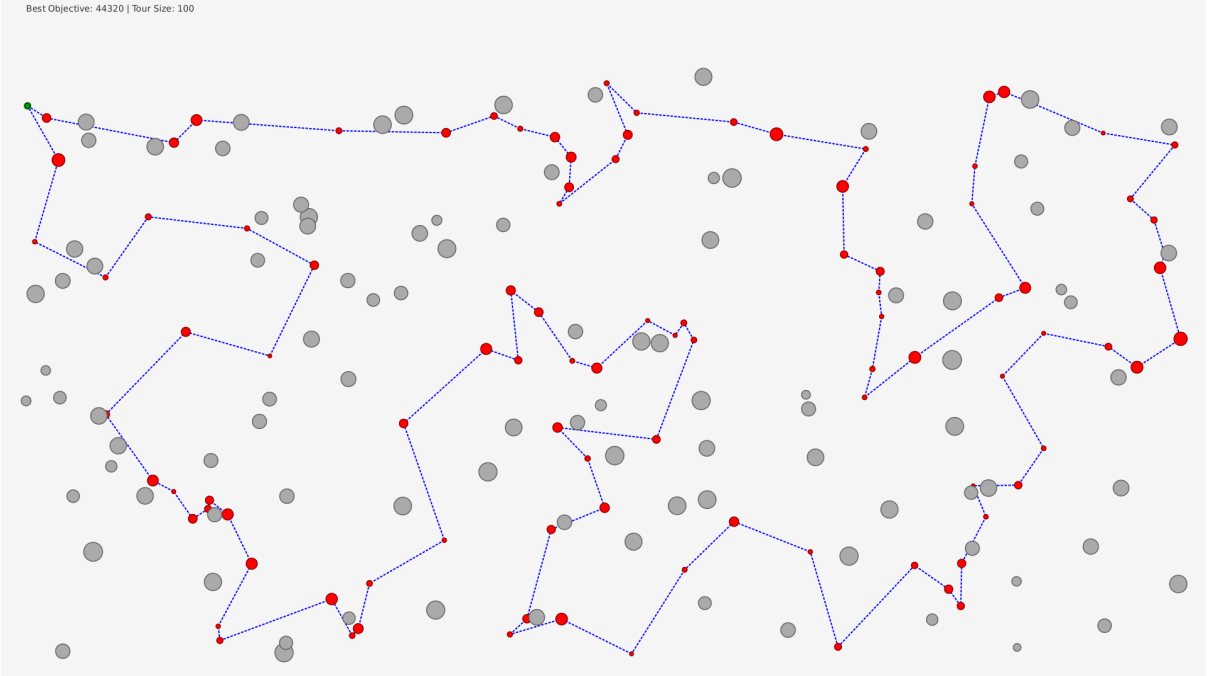
    - Instance B



- **Greedy LS 2-opt, greedy init**
    - Instance A



    - Instance B



Best Solutions

The best solutions were checked with the solution checker.

- **Steepest LS swap, rand init**
    - Instance A

93, 108, 18, 193, 139, 118, 51, 151, 162, 123, 35, 84, 112, 4, 190, 10, 177, 127, 70, 135, 154, 101, 97, 26, 100, 53, 158, 180, 121, 124, 148, 94, 63, 79, 80, 176, 137, 9, 62, 102, 49, 178, 106, 144, 14, 138, 165, 40, 185, 52, 152, 1, 2, 82, 129, 55, 57, 92, 78, 145, 196, 90, 81, 31, 56, 113, 175, 171, 16, 25, 44, 120, 75, 86, 133, 59, 181, 160, 116, 65, 47, 184, 54, 48, 34, 146, 22, 159, 41, 96, 5, 42, 43, 131, 149, 115, 46, 0, 143, 117

    - Instance B

35, 109, 0, 29, 144, 160, 33, 138, 182, 112, 151, 198, 117, 193, 31, 54, 73, 136, 142, 78, 175, 80, 190, 45, 5, 36, 141, 187, 127, 89, 103, 163, 153, 81, 82, 158, 121, 131, 1, 27, 38, 63, 135, 177, 61, 91, 77, 194, 166, 86, 95, 185, 94, 47, 148, 20, 28, 140, 183, 152, 155, 3, 15, 145, 43, 11, 104, 21, 25, 51, 90, 122, 107, 40, 133, 10, 147, 70, 13, 195, 168, 132, 169, 188, 6, 134, 139, 8, 111, 159, 124, 62, 18, 34, 55, 128, 176, 113, 106, 143

- **Steepest LS swap, greedy init**
    - Instance A

108, 69, 18, 159, 22, 146, 181, 34, 160, 48, 54, 177, 184, 84, 4, 112, 35, 131, 149, 65, 116, 43, 42, 5, 41, 193, 139, 68, 46, 115, 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, 179, 92, 129, 57, 55, 52, 185, 40, 196, 81, 90, 165, 106, 178, 14, 49, 102, 144, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 89, 183, 143, 0, 117, 93, 140

    - Instance B

131, 121, 51, 90, 147, 6, 188, 169, 132, 13, 195, 168, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 36, 61, 21, 82, 8, 104, 33, 160, 0, 35, 109, 29, 11, 138, 182, 25, 177, 5, 78, 175, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 38, 135, 63, 100, 40, 107, 122

- **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, greedy init**
    - Instance A

31, 56, 113, 175, 171, 16, 25, 44, 120, 92, 57, 129, 2, 75, 101, 1, 152, 97, 26, 100, 86, 53, 180, 154, 135, 70, 127, 123, 162, 151, 133, 79, 63, 94, 80, 176, 51, 118, 59, 65, 116, 43, 42, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 181, 146, 22, 18, 108, 69, 159, 193, 41, 139, 115, 46, 68, 140, 93, 117, 0, 143, 183, 89, 23, 137, 186, 114, 15, 148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 185, 40, 165, 90, 81, 196, 179, 145, 78

    - Instance B

135, 63, 100, 40, 107, 133, 10, 147, 6, 188, 169, 132, 13, 195, 168, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 36, 61, 21, 82, 8, 104, 111, 35, 109, 0, 29, 160, 33, 11, 139, 138, 182, 25, 177, 5, 78, 175, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 1, 131, 121, 51, 90, 122

- **Greedy LS swap, rand init**
    - Instance A

113, 31, 196, 81, 90, 165, 119, 40, 185, 106, 186, 89, 183, 117, 68, 139, 41, 181, 34, 160, 48, 54, 184, 84, 112, 127, 133, 162, 123, 4, 190, 10, 177, 35, 131, 149, 47, 65, 116, 46, 0, 143, 137, 176, 80, 124, 94, 122, 79, 151, 118, 115, 108, 18, 22, 146, 159, 193, 5, 42, 43, 59, 51, 148, 9, 62, 144, 14, 102, 49, 178, 57, 92, 120, 78, 145, 179, 52, 55, 129, 2, 152, 1, 101, 97, 26, 100, 121, 63, 135, 70, 154, 180, 53, 86, 75, 44, 16, 171, 175

    - Instance B

70, 3, 155, 184, 152, 170, 55, 18, 143, 82, 61, 36, 142, 45, 136, 190, 80, 162, 175, 78, 177, 21, 87, 111, 159, 106, 124, 166, 194, 176, 113, 103, 163, 89, 127, 165, 187, 153, 8, 104, 121, 198, 117, 193, 31, 54, 73, 173, 25, 138, 33, 49, 11, 139, 43, 168, 195, 145, 13, 132, 169, 6, 147, 51, 98, 118, 182, 144, 160, 29, 0, 109, 35, 34, 62, 128, 86, 185, 179, 94, 47, 148, 20, 28, 140, 183, 95, 81, 77, 141, 5, 1, 38, 135, 63, 40, 107, 122, 90, 188

- **Greedy LS swap, greedy init**
    - Instance A

108, 69, 18, 159, 22, 146, 181, 34, 160, 48, 54, 177, 184, 84, 4, 112, 35, 131, 149, 65, 116, 43, 42, 5, 41, 193, 139, 68, 46, 115, 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, 179, 92, 129, 57, 55, 52, 185, 40, 196, 81, 90, 165, 106, 178, 14, 49, 102, 144, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 89, 183, 143, 0, 117, 93, 140

    - Instance B

131, 121, 51, 90, 147, 6, 188, 169, 132, 13, 195, 168, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 36, 61, 21, 82, 8, 104, 33, 160, 0, 35, 109, 29, 11, 138, 182, 25, 177, 5, 78, 175, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 38, 135, 63, 40, 107, 133, 122

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

59, 65, 116, 43, 42, 41, 193, 159, 69, 108, 18, 22, 146, 34, 160, 54, 177, 10, 184, 84, 112, 127, 123, 162, 135, 70, 154, 180, 158, 53, 136, 63, 79, 133, 151, 51, 176, 80, 122, 94, 124, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 129, 92, 82, 120, 44, 25, 78, 16, 171, 175, 113, 56, 31, 145, 179, 196, 81, 90, 165, 119, 40, 185, 57, 55, 52, 106, 178, 3, 49, 102, 14, 144, 62, 9, 148, 137, 23, 186, 89, 183, 143, 0, 117, 93, 68, 46, 139, 115, 118

    - Instance B

138, 182, 11, 139, 43, 168, 195, 126, 13, 145, 15, 3, 70, 161, 132, 169, 188, 6, 147, 191, 90, 51, 98, 74, 118, 121, 131, 122, 135, 63, 38, 27, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 175, 78, 5, 177, 36, 61, 21, 87, 8, 104, 56, 144, 111, 82, 141, 77, 81, 153, 163, 89, 103, 113, 180, 176, 194, 166, 86, 95, 130, 99, 22, 185, 179, 94, 47, 148, 60, 20, 28, 149, 140, 183, 152, 34, 55, 18, 83, 62, 124, 106, 143, 35, 109, 0, 29, 160, 33

- **Greedy LS 2-opt, greedy init**
    - Instance A

31, 56, 113, 175, 171, 16, 25, 44, 120, 2, 75, 101, 1, 152, 97, 26, 100, 86, 53, 180, 154, 135, 70, 127, 123, 162, 151, 133, 79, 63, 94, 80, 176, 51, 118, 59, 65, 116, 43, 42, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 181, 146, 22, 18, 108, 69, 159, 193, 41, 139, 115, 46, 68, 140, 93, 117, 0, 143, 183, 89, 23, 137, 186, 114, 15, 148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 185, 40, 165, 90, 81, 196, 179, 57, 129, 92, 145, 78

    - Instance B

40, 107, 133, 10, 147, 6, 188, 169, 132, 13, 195, 168, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 36, 61, 21, 82, 8, 104, 111, 35, 109, 0, 29, 160, 33, 11, 139, 138, 182, 25, 177, 5, 78, 175, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 1, 131, 121, 51, 90, 122, 135, 63, 100

Conclusions

- Local search methods improved solution quality compared to baseline heuristics. Among the tested variants, the Steepest Local Search with 2-opt and greedy initialization achieved the best overall performance, producing the lowest objective values for both instances.
- The 2-opt operator outperformed swap in optimizing routes, and greedy initialization led to better results than random initialization. While Greedy Local Search was faster, the Steepest variant resulted in better solution quality.
- The Local Search methods improve the random solution dramatically, but they also take substantitailly more time then LS on the greedy initial solutions, which makes sense since they have less work to do.
- Since both of the LS methods still achieve worse scores with the random initializations we can clearly see that the solution space has many local minima.