Evolutionary Computation
Greedy Heuristics 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

---

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)

N - # of nodes

K - # of nodes to be selected

1. **Random algorithm**

   A simple non-deterministic baseline.

   1. Initialize a list of all node indices (0 to N-1).
   2. Randomly shuffle this list.
   3. Select the first K elements from the shuffled list to form the tour.

2. **Nearest Neighbour end-only**

Initialize `n ← size(d)`.

Initialize an empty list `path` and a boolean array `used` of size `n` ← false.

Add `start_node` to `path` and set `used[start_node] ← true`.

While |path| < k **do**
    a. Set `best_node ← -1` and `best_value ← ∞`.
    b. Let `current ← last element of path`.
    c. **For each** node `cand` from 0 to `n - 1` **do**
        If `used[cand] = false` **then**
            Compute `val ← d[current][cand] + nodes[cand].cost`.
            If `val < best_value` **then**
                `best_value ← val`
                `best_node ← cand`
    d. **If** `best_node = -1`, **then** break.
    e. Add `best_node` to `path` and set `used[best_node] ← true`.

Return `path`.

3. **Nearest Neighbour all-positions**

Initialize an empty list `path` and a boolean array `used` of size `n` ← false.

Add `start_node` to `path` and set `used[start_node] ← true`.

While |path| < k **do**

a. Set `next_node ← -1`, `best_position ← -1`, and `best_value ← ∞`.

b. **For each** unused node `cand` **do**

Compute `val ← d[cand][path[0]] + nodes[cand].cost`.

If `val < best_value` **then**

`best_value ← val`

`next_node ← cand`

c. **For** each index `i` in `path` **do**

Let `after ← path[i]` and `before ← path[(i + 1) mod |path|]`.

For each unused node `cand` **do**

Compute `val ← d[after][cand] + nodes[cand].cost + d[cand][before] - d[after][before]`.

If `val < best_value` **then**

`best_value ← val`

`next_node ← cand`

`best_position ← i`

d. Insert `next_node` into `path` after position `best_position`.

e. Set `used[next_node] ← true`.

Return `path`.

4. **Greedy Cycle**

Initialize `n ← size(d)`.

Initialize an empty list `cycle` and a boolean array `used` of size `n` ← false.

Add `start_node` to `cycle` and set `used[start_node] ← true`.

Select the second node:
- a. Set `best_second ← -1` and `best_value ← ∞`.
- b. **For each** unused node `cand` **do**
  - Compute `val ← d[start_node][cand] + nodes[cand].cost`.
  - **If** `val < best_value` **then**
    - `best_value ← val`
    - `best_second ← cand`
- c. Add `best_second` to `cycle` and set `used[best_second] ← true`.

While |cycle| < k **do**
- a. Set `next_node ← -1`, `best_position ← 0`, and `best_value ← ∞`.
- b. Let `c_size ← |cycle|`.
- c. **For** each index `i` in `cycle` **do**
  - Let `after ← cycle[i]` and `before ← cycle[(i + 1) mod c_size]`.
  - Set `local_best ← ∞`, `local_node ← -1`.
  - **For each** unused node `cand` **do**
    - Compute `val ← d[after][cand] + nodes[cand].cost + d[cand][before]`.
    - **If** `c_size > 2`, then `val ← val - d[after][before]`.
    - **If** `val < local_best` **then**
      - `local_best ← val`
      - `local_node ← cand`
  - **If** `local_best < best_value` **then**
    - `best_value ← local_best`
    - `next_node ← local_node`
    - `best_position ← i`
- d. Insert `next_node` into `cycle` after position `best_position`.
- e. Set `used[next_node] ← true`.

Return `cycle`.

5. Nearest Neighbour all-positions with weighted 2-regret

Initialize `n ← size(d)`.

Initialize an empty list `path` and a boolean array `used` of size `n` ← false.

Add `start_node` to `path` and set `used[start_node] ← true`.

While |path| < k do
- a. Initialize an empty list `ranking`.
- b. For each unused node `i` do
  - i. Set `best_first_val ← d[i][path[0]] + nodes[i].cost`, `best_first_pos ← 0`.
  - ii. Set `best_second_val ← ∞`, `best_second_pos ← 0`.
  - iii. Let `c_size ← |path|`.
  - iv. For each position `pos` from 1 to `c_size - 1` do
    - Compute `val ← d[path[pos - 1]][i] + d[path[pos]][i] - d[path[pos - 1]][path[pos]] + nodes[i].cost`.
    - If `val < best_first_val` then
      - Set `best_second_val ← best_first_val`, `best_second_pos ← best_first_pos`, `best_first_val ← val`, `best_first_pos ← pos`.
    - Else if `val < best_second_val` then
      - Set `best_second_val ← val`, `best_second_pos ← pos`.
  - v. Compute `end_val ← d[path[c_size - 1]][i] + nodes[i].cost`.
  - vi. If `end_val < best_first_val` then
    - Set `best_second_val ← best_first_val`, `best_second_pos ← best_first_pos`, `best_first_val ← end_val`, `best_first_pos ← c_size`.
    - Else if `end_val < best_second_val` then
      - Set `best_second_val ← end_val`, `best_second_pos ← c_size`.
  - vii. Add entry `{i, best_first_pos, best_first_val, best_second_pos, best_second_val}` to `ranking`.
- c. Set `best_score ← -∞`, `best_node ← 0`, `best_pos ← 0`.
- d. For each entry in `ranking` do
  - Compute `score ← w1 × (entry.second_val - entry.first_val) - w2 × entry.first_val`.
  - If `score > best_score` then
    - `best_score ← score`, `best_node ← entry.node`, `best_pos ← entry.first_pos`.
- e. Insert `best_node` into `path` at position `best_pos`.
- f. Set `used[best_node] ← true`.

Return `path`.

6. Greedy Cycle with weighted 2-regret

Initialize `n ← size(d)`.

Initialize an empty list `cycle` and a boolean array `used` of size `n` ← false.

Add `start_node` to `cycle` and set `used[start_node] ← true`.

Select the second node:
- a. Set `best_second ← -1` and `best_value ← ∞`.
- b. For each unused node `cand` do
  - Compute `val ← d[start_node][cand] + nodes[cand].cost`.
  - If `val < best_value` then
    - `best_value ← val`
    - `best_second ← cand`.
- c. Add `best_second` to `cycle` and set `used[best_second] ← true`.

While |cycle| < k do
- a. Initialize an empty list `ranking`.
- b. For each unused node `i` do
  - i. Set `best_first_val ← ∞`, `best_second_val ← ∞`.
  - ii. Set `best_first_pos ← 0`, `best_second_pos ← 0`.
  - iii. Let `c_size ← |cycle|`.
  - iv. For each position `pos` from 1 to `c_size` do
    - Compute
    - `val ← d[cycle[pos - 1]][i] + d[cycle[pos mod c_size]][i] + nodes[i].cost`.
    - If `c_size > 2`, then
      - `val ← val - d[cycle[pos - 1]][cycle[pos mod c_size]]`.
    - If `val < best_first_val` then
      - Set `best_second_val ← best_first_val`, `best_second_pos ← best_first_pos`,
      - `best_first_val ← val`, `best_first_pos ← pos`.
    - Else if `val < best_second_val` then
      - Set `best_second_val ← val`, `best_second_pos ← pos`.
  - v. Add entry `{i, best_first_pos, best_first_val, best_second_pos, best_second_val}` to `ranking`.
- c. Set `best_score ← -∞`, `best_node ← 0`, `best_pos ← 0`.
- d. For each entry in `ranking` do
  - Compute
  - `score ← w1 × (entry.second_val - entry.first_val) - w2 × entry.first_val`.
  - If `score > best_score` then
    - `best_score ← score`, `best_node ← entry.node`, `best_pos ← entry.first_pos`.
- e. Insert `best_node` into `cycle` at position `best_pos`.
- f. Set `used[best_node] ← true`.

Return `cycle`.

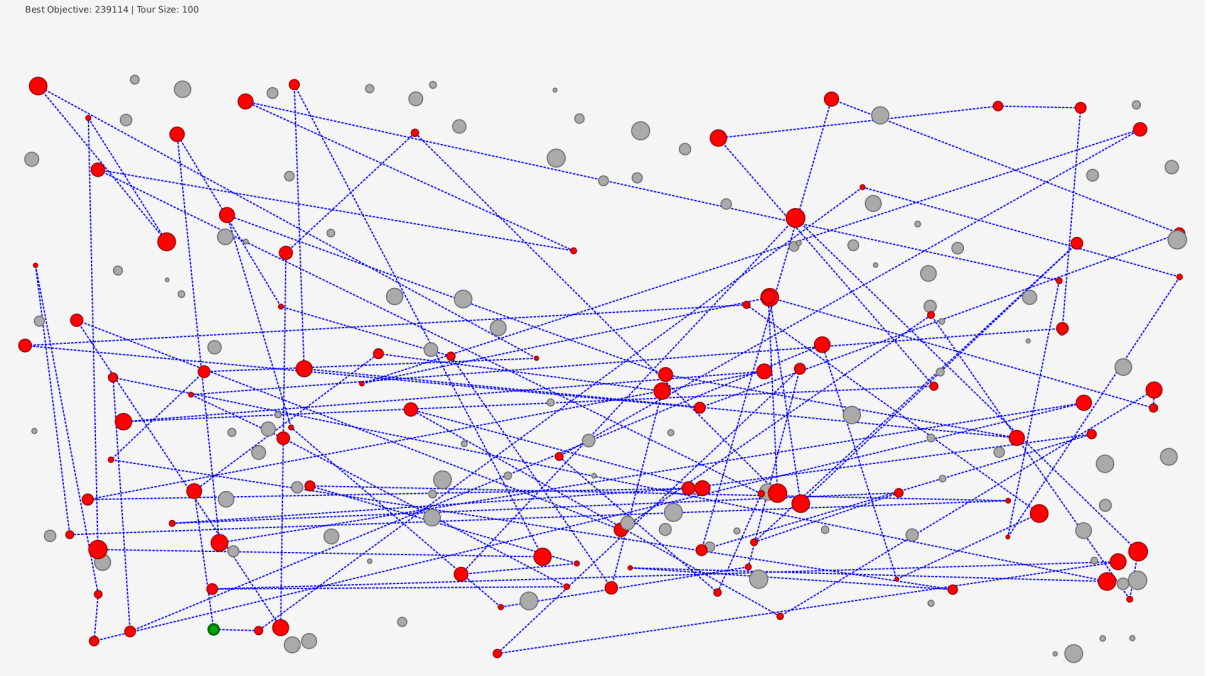
Results and Analysis

Algorithm	Instance A			Instance B		
	Min	Max	Avg	Min	Max	Avg
Random	239114	291474	264152	185581	238526	212540
NN end-only	83182	89433	85108.5	52319	59030	54390.4
NN all-pos	71695	75953	73302.4	44242	57283	48498.9
Greedy Cycle	71488	74410	72617.6	48765	57324	51339.5
NN all-pos 2-regret	108151	124921	117138	69933	80278	74444.5
Greedy Cycle 2-regret	105692	126951	115579	67809	78406	72740
NN all-pos 2-regret weighted (0.5, 0.5)	70010	75452	72401.2	44891	55247	47664.5
Greedy Cycle 2-regret weighted (0.5, 0.5)	71108	73395	72129.7	47144	55700	50897.1

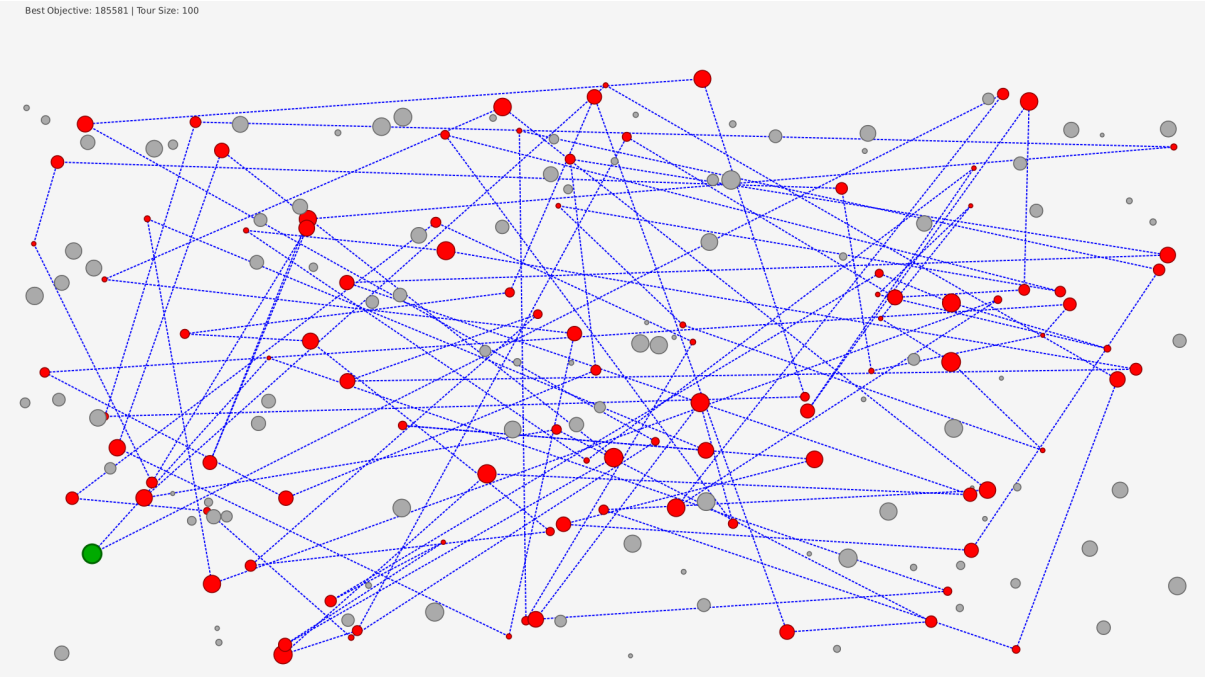
Visual Comparisons (Visual Comparision)

Size of the dot corresponds to it's cost (the bigger it is the bigger the cost), and the green dot is the starting node.

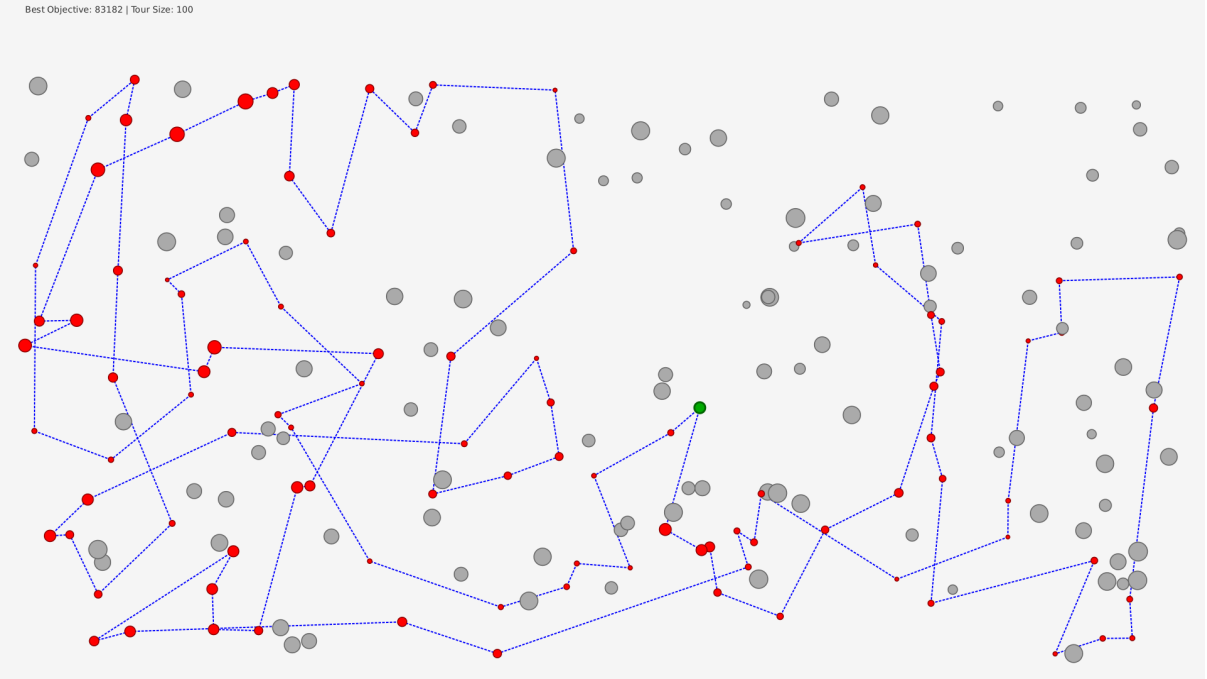
- Random algorithm
    - Instance A:



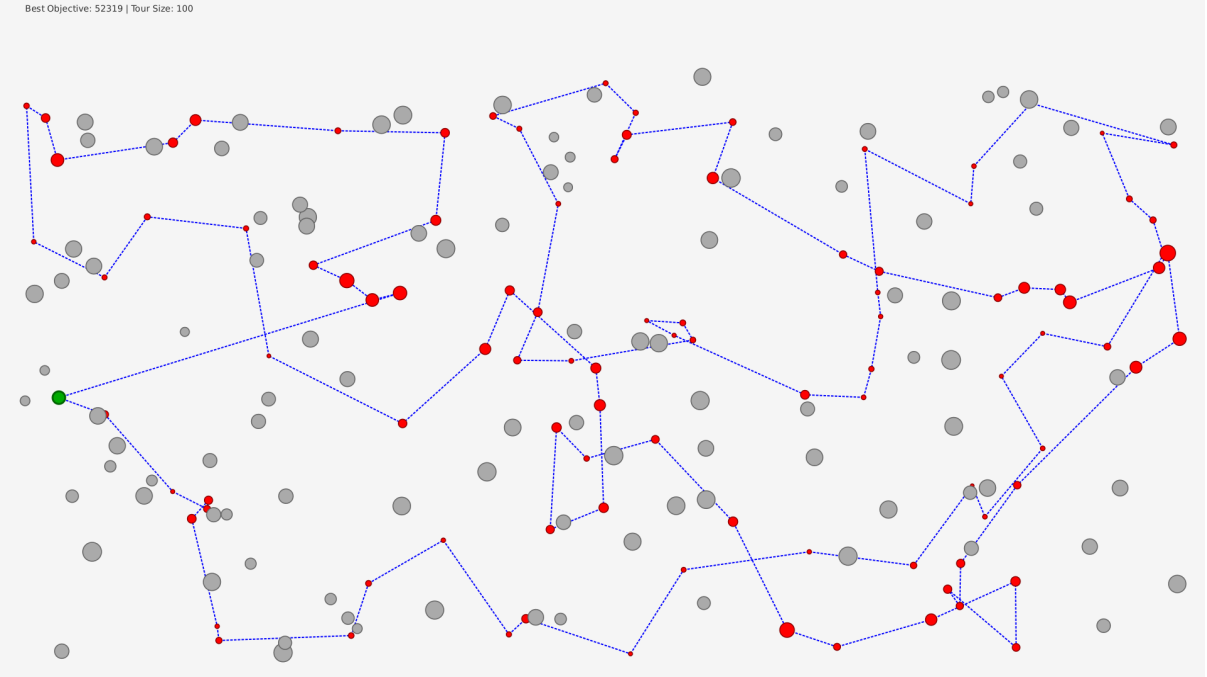
    - Instance B:



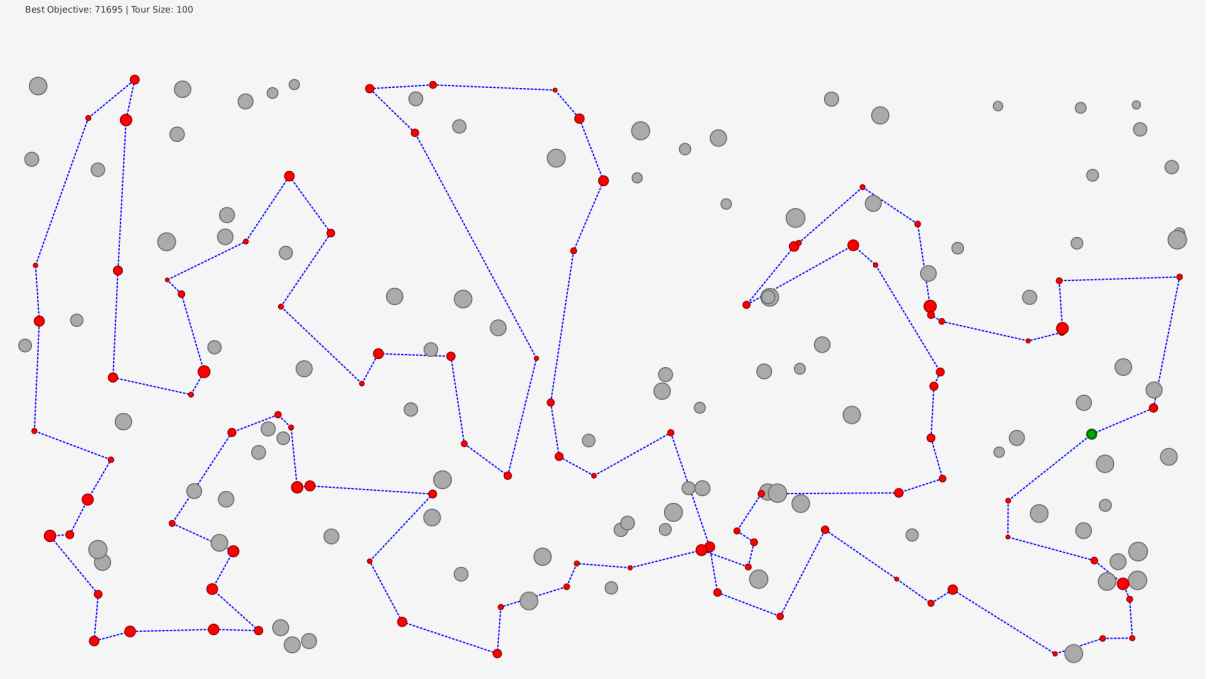
- Nearest Neighbour end-only
    - Instance A:



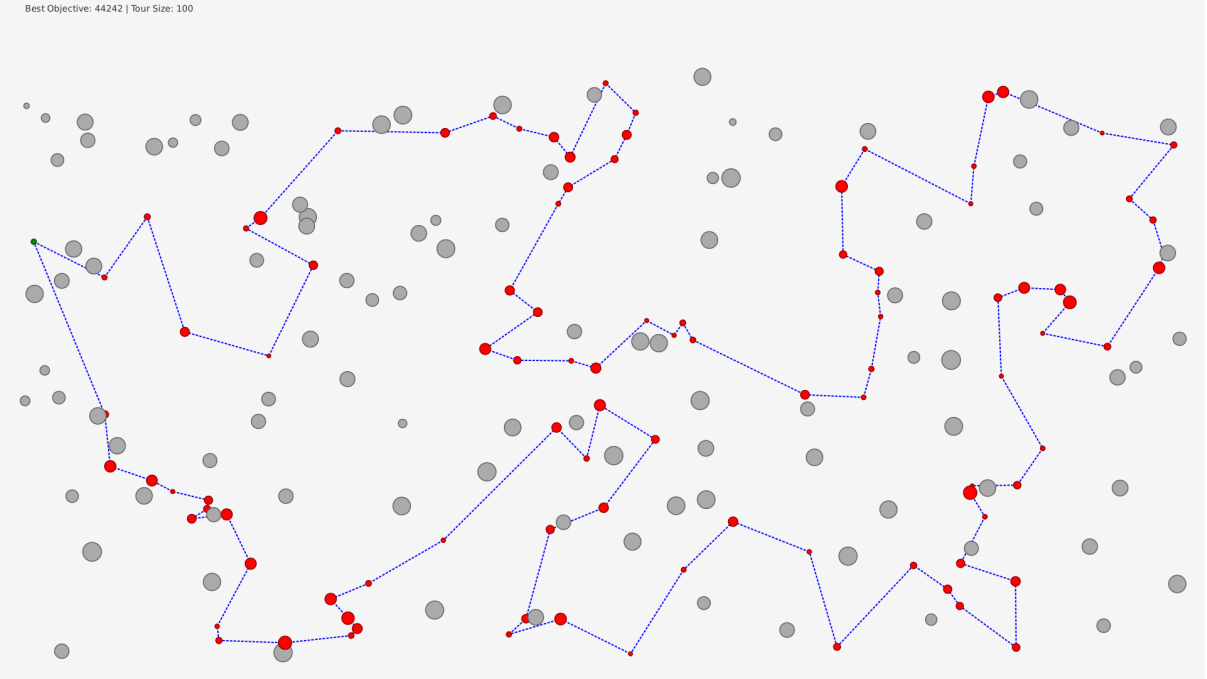
    - Instance B:



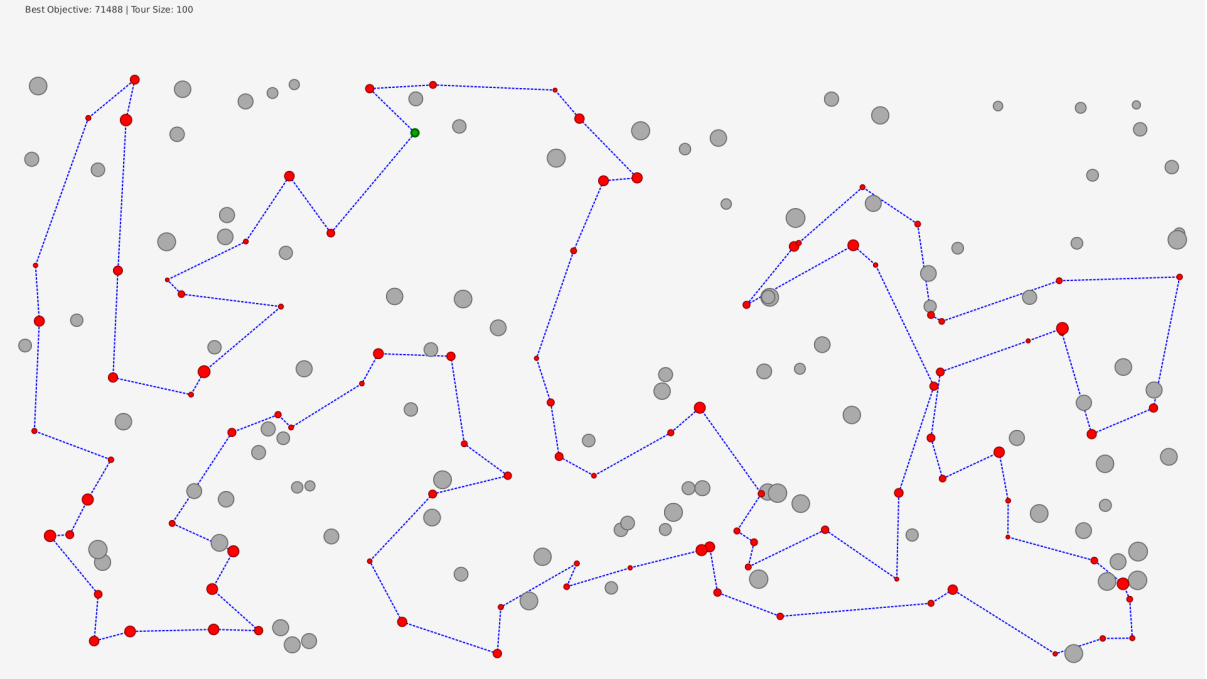
- Nearest Neighbour all-positions
    - Instance A:



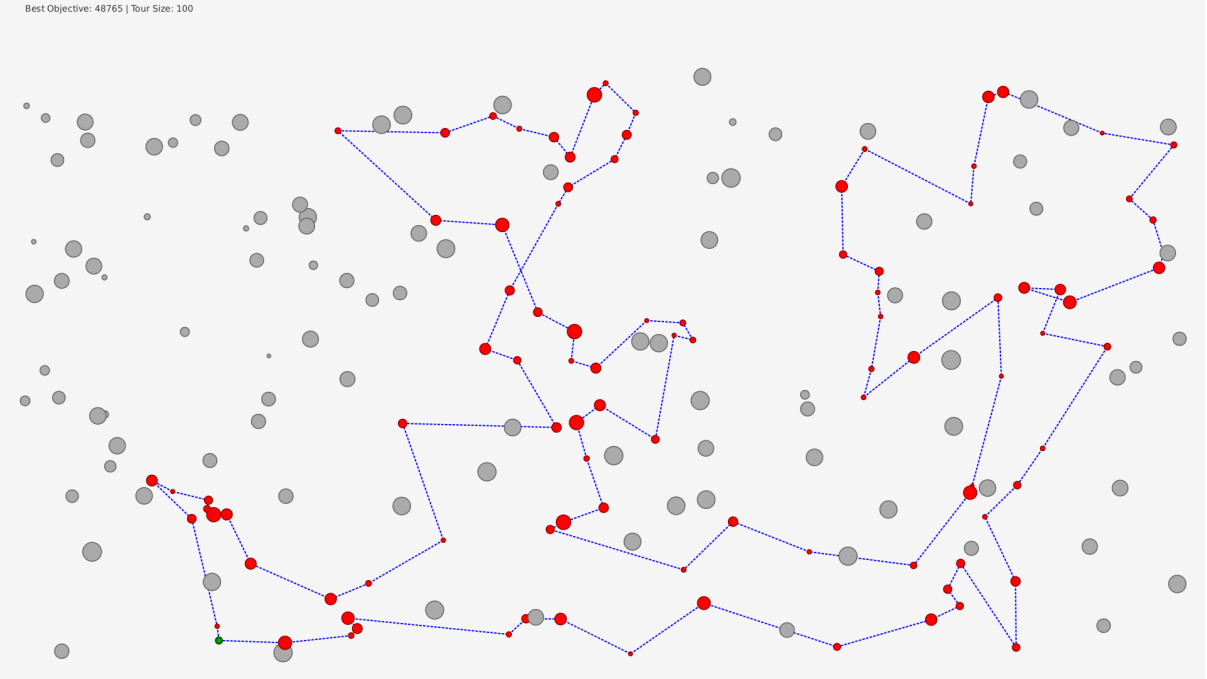
    - Instance B:



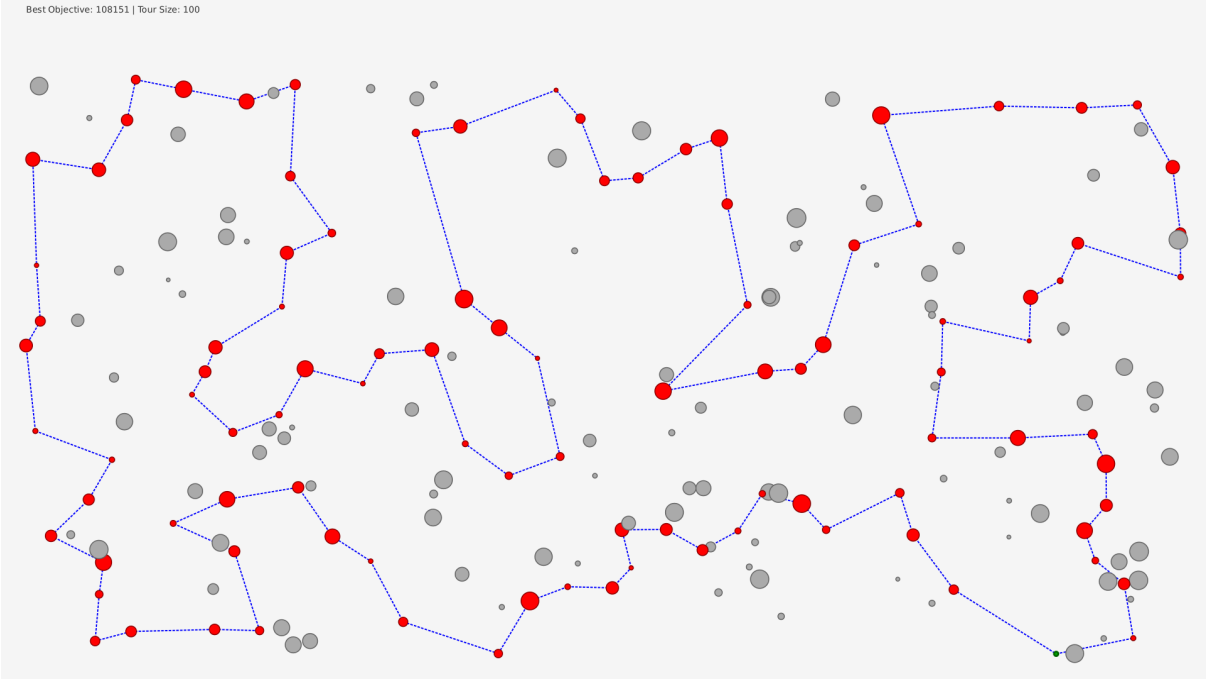
- Greedy Cycle
    - Instance A:



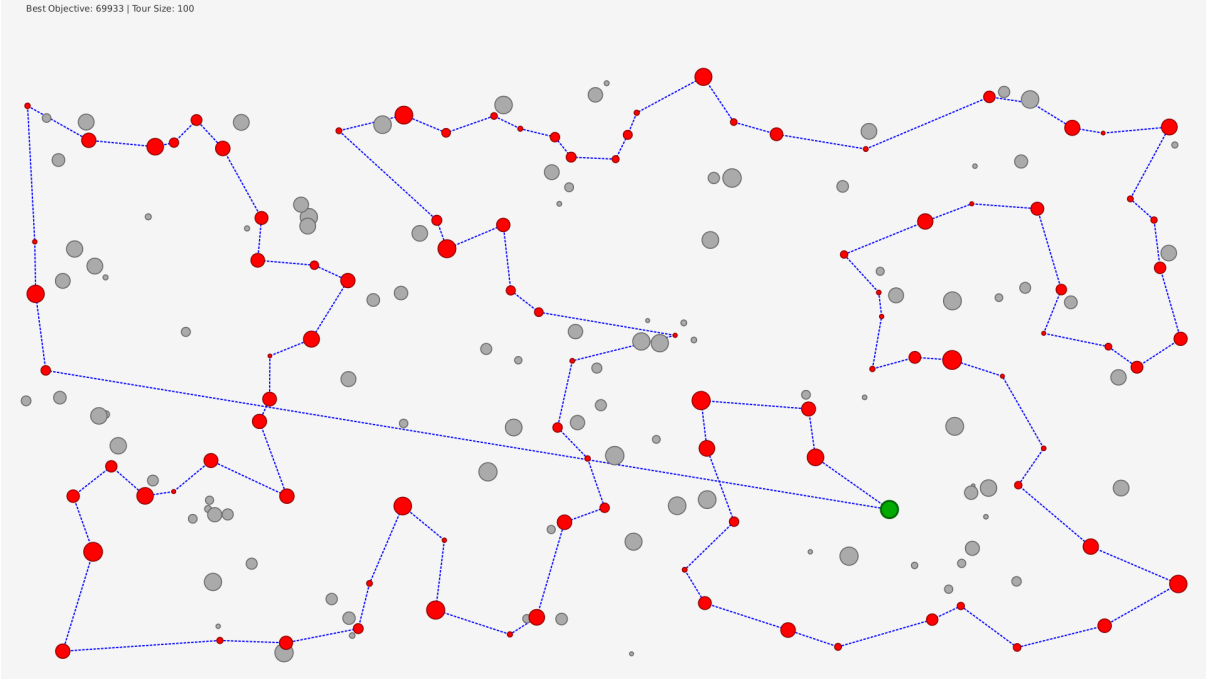
    - Instance B:



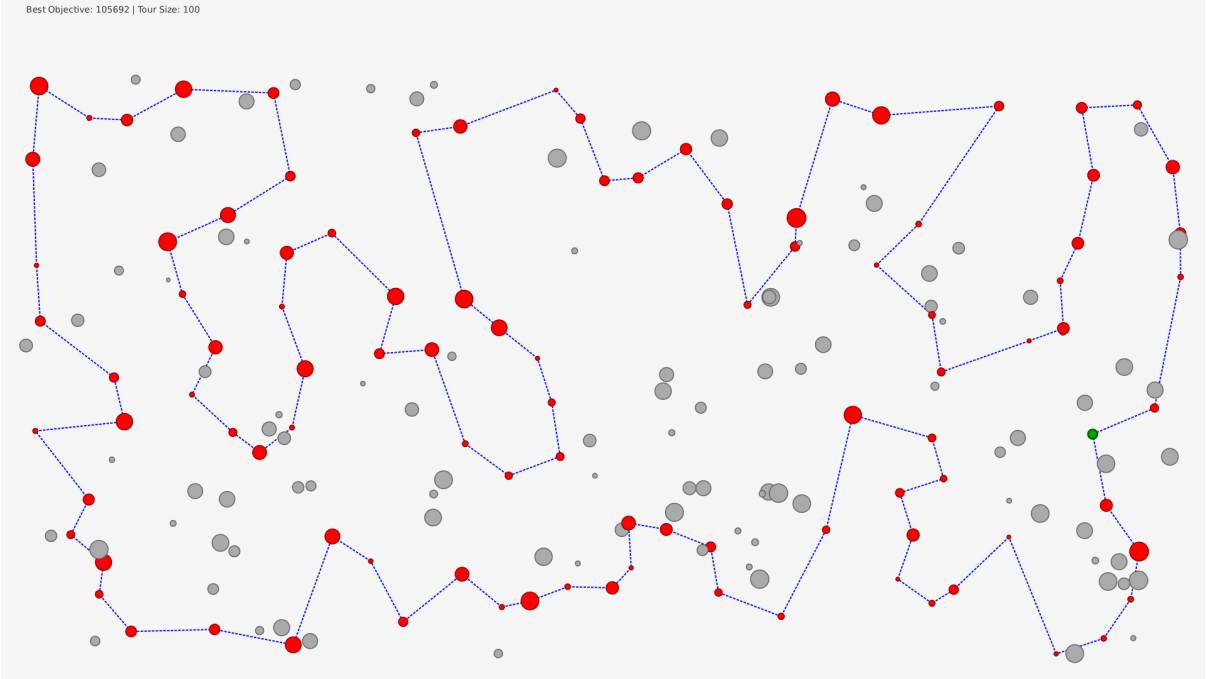
- Nearest Neighbour all-positions with 2-regret
    - Instance A:



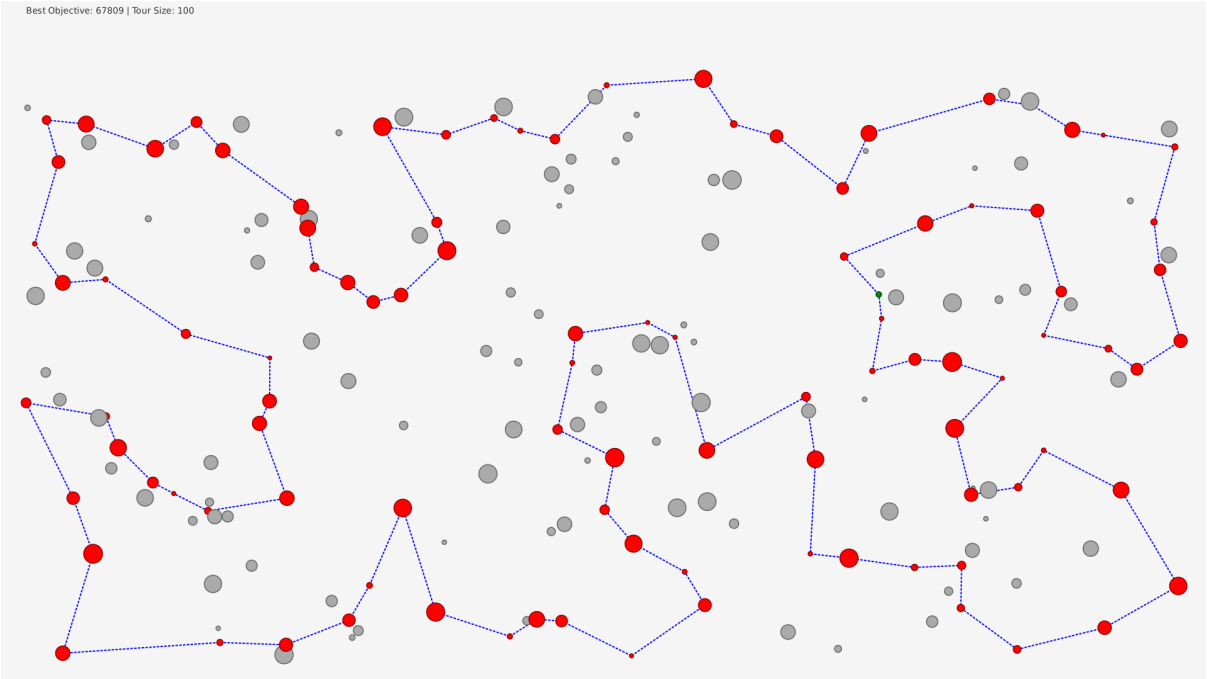
    - Instance B:



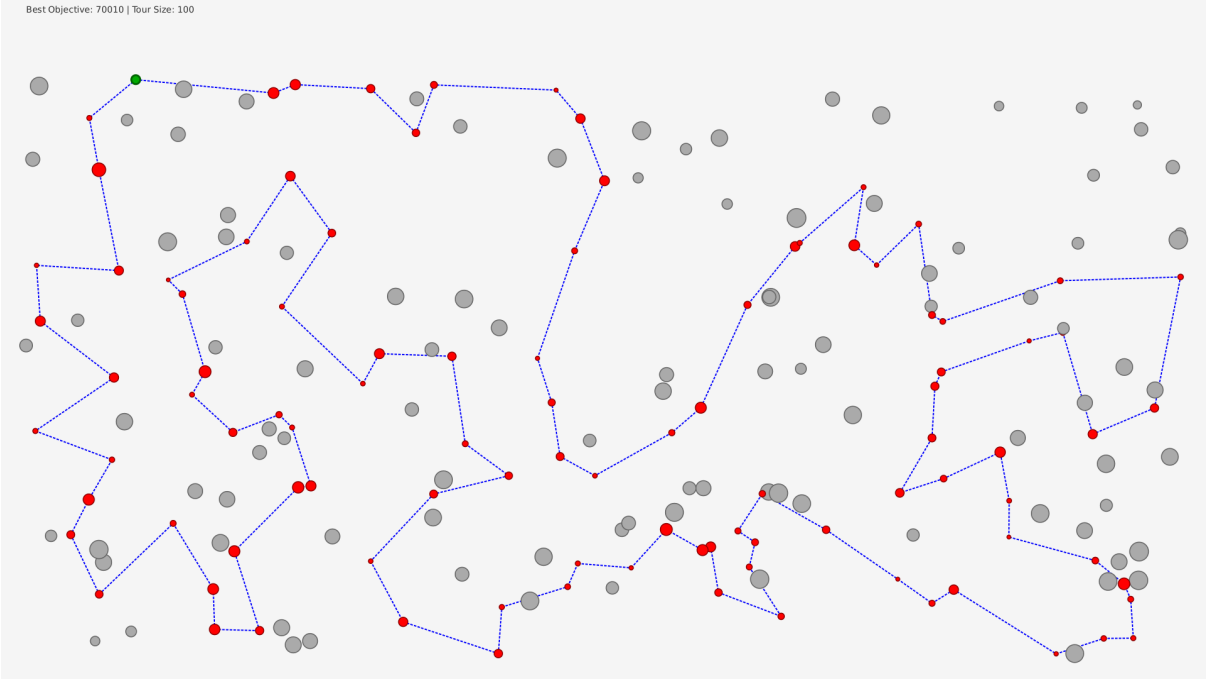
- Greedy Cycle with 2-regret
    - Instance A:



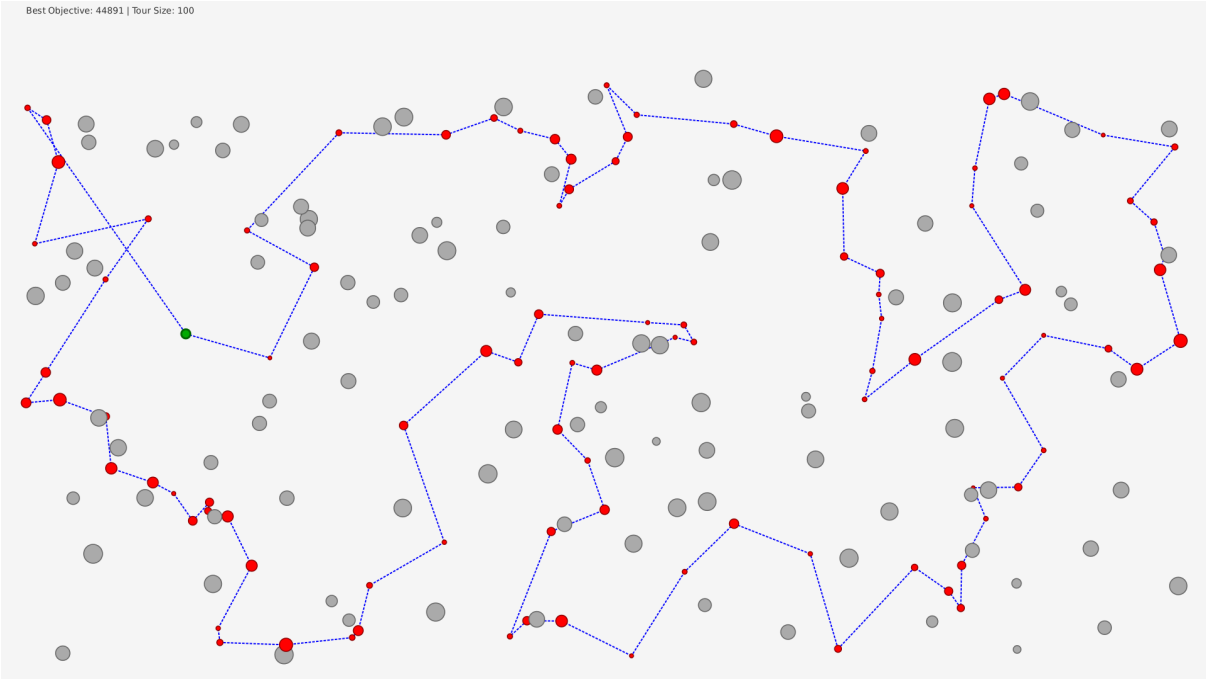
    - Instance B:



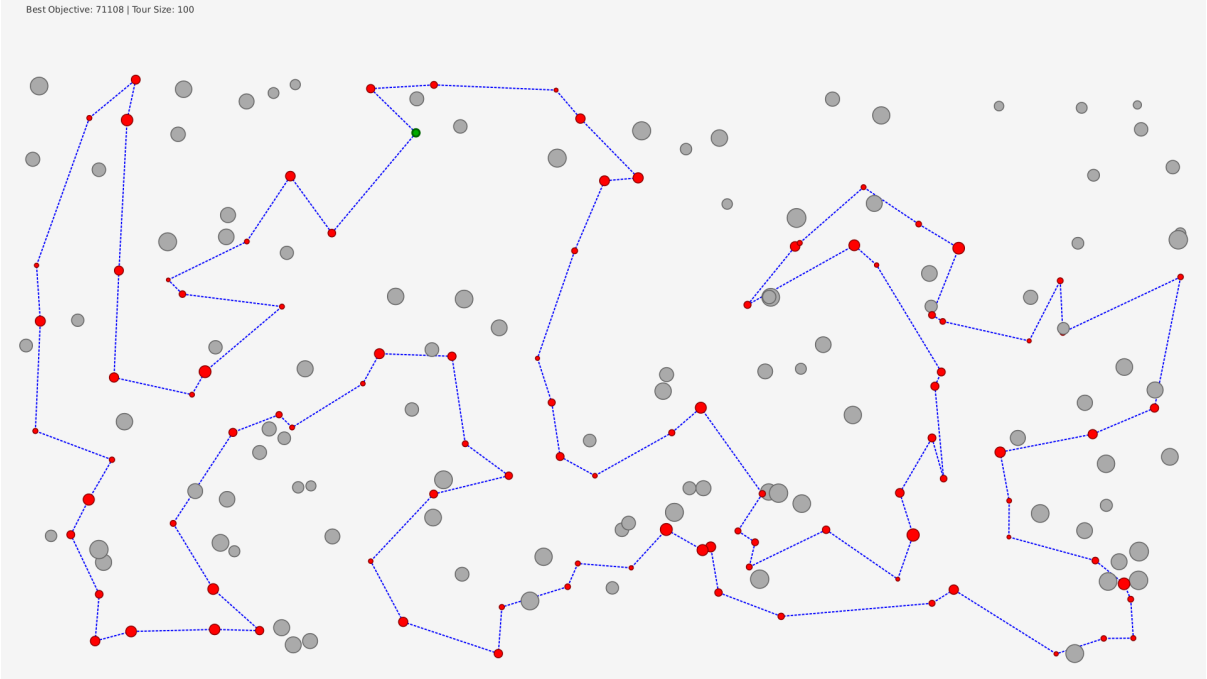
- Nearest Neighbour all-positions with 2-regret weighted
    - Instance A:



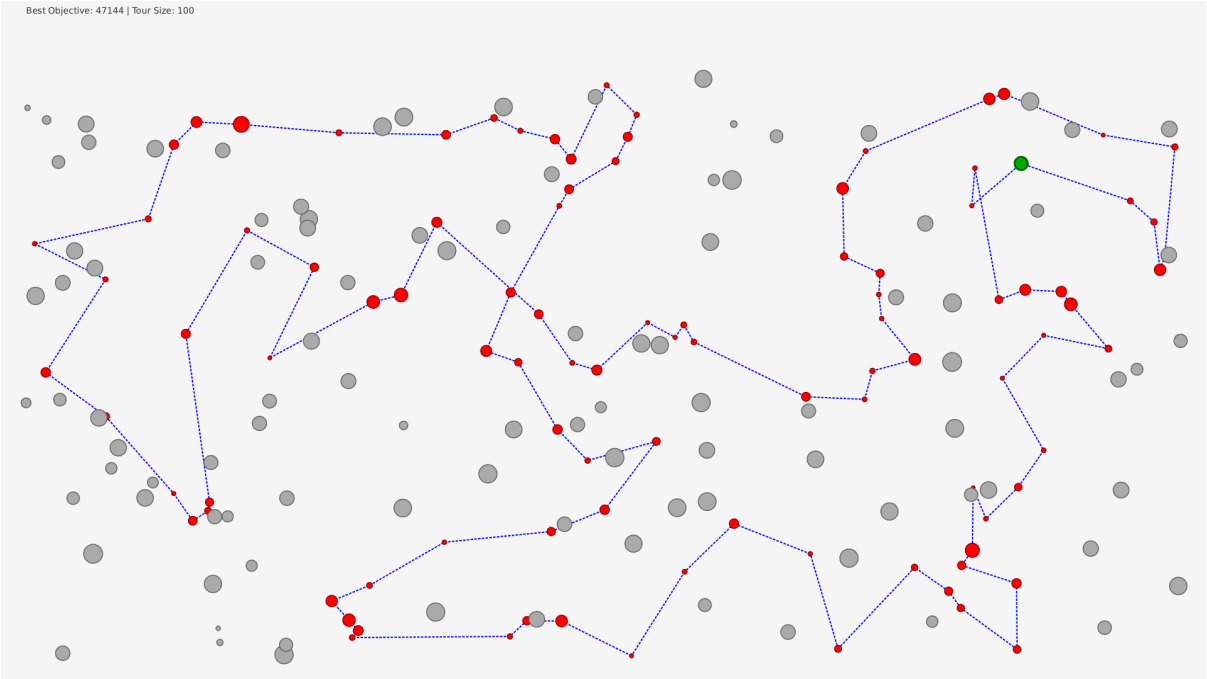
    - Instance B:



- Greedy Cycle with 2-regret weighted
    - Instance A:



    - Instance B:



Best Solutions

The best solutions were checked with the solution checker.

- Random algorithm
    - Instance A: 34, 30, 144, 54, 89, 103, 181, 97, 135, 76, 64, 2, 171, 43, 85, 136, 66, 156, 141, 51, 197, 158, 41, 101, 42, 88, 179, 154, 84, 77, 177, 53, 195, 3, 60, 188, 25, 165, 132, 46, 105, 20, 199, 118, 166, 124, 86, 70, 80, 139, 63, 92, 9, 187, 178, 27, 15, 109, 137, 98, 58, 18, 56, 31, 26, 95, 28, 57, 16, 184, 186, 29, 94, 65, 142, 12, 190, 23, 78, 35, 194, 10, 7, 185, 91, 38, 82, 176, 1, 167, 160, 48, 62, 110, 127, 145, 151, 116, 161, 11
    - Instance B: 196, 149, 50, 95, 105, 122, 41, 25, 119, 87, 26, 66, 169, 61, 35, 134, 42, 31, 175, 111, 17, 84, 143, 1, 10, 60, 67, 151, 120, 198, 63, 100, 170, 124, 185, 22, 38, 173, 160, 13, 154, 98, 180, 171, 46, 78, 15, 83, 130, 101, 159, 140, 8, 90, 2, 172, 158, 11, 136, 21, 116, 131, 139, 161, 146, 165, 104, 30, 70, 52, 137, 82, 88, 65, 135, 49, 36, 24, 178, 68, 162, 177, 45, 55, 181, 179, 18, 183, 62, 110, 166, 109, 168, 99, 6, 81, 37, 79, 89, 121

- Nearest Neighbour end-only
    - Instance A: 124, 94, 63, 53, 180, 154, 135, 123, 65, 116, 59, 115, 139, 193, 41, 42, 160, 34, 22, 18, 108, 69, 159, 181, 184, 177, 54, 30, 48, 43, 151, 176, 80, 79, 133, 162, 51, 137, 183, 143, 0, 117, 46, 68, 93, 140, 36, 163, 199, 146, 195, 103, 5, 96, 118, 149, 131, 112, 4, 84, 35, 10, 190, 127, 70, 101, 97, 1, 152, 120, 78, 145, 185, 40, 165, 90, 81, 113, 175, 171, 16, 31, 44, 92, 57, 106, 49, 144, 62, 14, 178, 52, 55, 129, 2, 75, 86, 26, 100, 121
    - Instance B: 16, 1, 117, 31, 54, 193, 190, 80, 175, 5, 177, 36, 61, 141, 77, 153, 163, 176, 113, 166, 86, 185, 179, 94, 47, 148, 20, 60, 28, 140, 183, 152, 18, 62, 124, 106, 143, 0, 29, 109, 35, 33, 138, 11, 168, 169, 188, 70, 3, 145, 15, 155, 189, 34, 55, 95, 130, 99, 22, 66, 154, 57, 172, 194, 103, 127, 89, 137, 114, 165, 187, 146, 81, 111, 8, 104, 21, 82, 144, 160, 139, 182, 25, 121, 90, 122, 135, 63, 40, 107, 100, 133, 10, 147, 6, 134, 51, 98, 118, 74

- Nearest Neighbour all-positions
    - Instance A: 196, 81, 90, 165, 119, 40, 185, 106, 178, 3, 14, 144, 62, 9, 148, 102, 49, 52, 55, 57, 92, 129, 152, 97, 1, 101, 100, 53, 180, 154, 135, 70, 127, 123, 162, 149, 131, 65, 116, 43, 184, 35, 84, 112, 4, 190, 10, 177, 30, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 41, 193, 139, 68, 46, 115, 59, 118, 51, 151, 133, 176, 0, 117, 143, 183, 89, 23, 137, 80, 79, 63, 94, 26, 86, 75, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145
    - Instance B: 63, 135, 122, 131, 121, 51, 90, 191, 147, 6, 188, 169, 132, 13, 70, 3, 15, 145, 195, 168, 139, 11, 182, 138, 33, 160, 29, 0, 109, 35, 143, 106, 124, 62, 18, 55, 34, 170, 152, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 179, 185, 22, 99, 130, 95, 86, 166, 194, 176, 180, 113, 103, 114, 137, 127, 89, 163, 187, 153, 81, 77, 141, 91, 36, 61, 21, 82, 111, 144, 8, 104, 177, 5, 45, 142, 78, 175, 162, 80, 190, 136, 73, 193, 31, 54, 117, 198, 156, 1

- Greedy Cycle
    - Instance A: 0, 117, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 152, 97, 1, 101, 2, 120, 129, 55, 49, 102, 148, 9, 62, 144, 14, 178, 106, 165, 90, 81, 196, 40, 119, 185, 52, 57, 92, 179, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 75, 86, 26, 100, 53, 154, 180, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 184, 35, 84, 112, 4, 190, 10, 177, 30, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 115, 41, 193, 139, 68, 46
    - Instance B: 80, 162, 175, 78, 142, 36, 61, 91, 141, 97, 187, 165, 127, 89, 103, 137, 114, 113, 194, 166, 179, 185, 99, 130, 22, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 86, 176, 180, 163, 153, 81, 77, 21, 87, 82, 8, 56, 144, 111, 0, 35, 109, 29, 160, 33, 49, 11, 43, 134, 147, 6, 188, 169, 132, 13, 161, 70, 3, 15, 145, 195, 168, 139, 182, 138, 104, 25, 177, 5, 45, 136, 73, 164, 31, 54, 117, 198, 193, 190

- Nearest Neighbour all-positions with 2-regret
    - Instance A: 16, 175, 56, 31, 38, 157, 17, 196, 91, 57, 52, 106, 185, 8, 165, 39, 90, 27, 71, 164, 7, 21, 132, 14, 102, 128, 167, 111, 130, 148, 15, 64, 114, 186, 23, 89, 183, 153, 0, 141, 66, 176, 79, 133, 151, 109, 118, 59, 197, 116, 43, 42, 5, 96, 115, 198, 46, 68, 93, 36, 67, 108, 69, 199, 20, 22, 146, 103, 34, 160, 48, 30, 104, 177, 10, 190, 4, 112, 35, 184, 166, 131, 24, 123, 127, 70, 6, 154, 158, 53, 136, 121, 100, 97, 152, 87, 2, 129, 82, 25
    - Instance B: 129, 119, 159, 37, 41, 81, 77, 97, 146, 187, 165, 127, 137, 75, 93, 76, 194, 166, 86, 110, 128, 124, 62, 18, 34, 174, 183, 9, 99, 185, 179, 172, 57, 66, 47, 148, 23, 20, 59, 28, 4, 152, 184, 155, 84, 3, 15, 145, 13, 132, 169, 188, 6, 150, 147, 134, 2, 43, 139, 11, 0, 33, 104, 8, 82, 87, 79, 36, 7, 177, 123, 5, 78, 162, 80, 108, 196, 42, 156, 30, 117, 151, 173, 19, 112, 121, 116, 98, 51, 125, 191, 178, 10, 133, 44, 72, 40, 63, 92, 38

- Greedy Cycle with 2-regret
    - Instance A: 196, 157, 188, 113, 171, 16, 78, 25, 44, 120, 82, 129, 92, 57, 172, 2, 75, 86, 26, 121, 182, 53, 158, 154, 6, 135, 194, 127, 123, 24, 156, 4, 190, 177, 104, 54, 48, 34, 192, 181, 146, 22, 20, 134, 18, 69, 67, 140, 68, 110, 142, 41, 96, 42, 43, 77, 65, 197, 115, 198, 46, 60, 118, 109, 151, 133, 79, 80, 176, 66, 141, 0, 153, 183, 89, 23, 186, 114, 15, 148, 9, 61, 73, 132, 21, 14, 49, 178, 52, 185, 119, 165, 39, 95, 7, 164, 71, 27, 90, 81
    - Instance B: 18, 34, 174, 183, 9, 99, 185, 179, 172, 57, 66, 47, 60, 20, 59, 28, 4, 53, 170, 184, 155, 84, 70, 132, 169, 188, 6, 192, 134, 2, 74, 118, 98, 51, 120, 71, 178, 10, 44, 17, 107, 100, 63, 102, 135, 131, 121, 112, 19, 173, 31, 117, 198, 24, 1, 27, 42, 196, 108, 80, 162, 142, 5, 123, 7, 36, 79, 91, 141, 97, 77, 58, 82, 68, 104, 33, 49, 29, 0, 41, 143, 119, 153, 186, 163, 103, 127, 137, 75, 93, 48, 166, 194, 180, 64, 86, 110, 128, 124, 62

- Nearest Neighbour all-positions with 2-regret weighted
    - Instance A: 108, 18, 199, 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, 168, 195, 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, 61, 36, 21, 82, 8, 104, 33, 160, 0, 35, 109, 29, 11, 138, 182, 25, 177, 5, 78, 175, 162, 80, 190, 136, 73, 31, 54, 193, 117, 198, 156, 1, 16, 27, 38, 135, 122, 63, 100, 107, 40

- Greedy Cycle with 2-regret weighted
    - Instance A: 0, 117, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 152, 97, 1, 101, 2, 120, 82, 129, 57, 92, 55, 52, 49, 102, 148, 9, 62, 144, 14, 138, 178, 106, 185, 165, 40, 90, 81, 196, 179, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 75, 86, 26, 100, 121, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 184, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 115, 41, 193, 139, 68, 46
    - Instance B: 199, 183, 140, 95, 130, 99, 22, 179, 185, 86, 166, 194, 113, 176, 26, 103, 114, 137, 127, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 175, 78, 142, 45, 5, 177, 21, 82, 111, 8, 104, 138, 182, 139, 168, 195, 145, 15, 3, 70, 13, 132, 169, 188, 6, 147, 115, 10, 133, 122, 63, 135, 38, 1, 117, 193, 31, 54, 131, 90, 51, 121, 118, 74, 134, 11, 33, 160, 29, 0, 109, 35, 143, 106, 124, 128, 62, 18, 55, 34, 170, 152, 4, 149, 28, 20, 60, 94, 66, 47, 148

Conclusions

- The Random algorithm serves only as a non-deterministic baseline, performing drastically worse than all other methods, with average objective values over 264,000 and 212,000 for Instances A and B, respectively.
- The basic greedy heuristics that allow for flexible insertion (NN all-positions and Greedy Cycle) significantly outperformed the constrained NN end-only method. For Instance A, the average result improved from 85,108.5 (NN end-only) to approximately 72,000 (insertion-based).
- The pure 2−regret variations (NN all-pos 2-regret and Greedy Cycle 2-regret) performed poorly, with average objective values much worse than the basic greedy methods (e.g., averages around 117,000 for Instance A). This suggests that prioritizing the difference between the two best moves (regret) over the absolute best cost-of-insertion leads to locally bad decisions.

- The weighted 2-regret approach, combining both regret and minimum cost, proved highly effective. The NN all-positions 2-regret weighted (0.5, 0.5) achieved the best overall performance for both instances, yielding the lowest minimum and lowest average objective values (Min: 70,010 for Instance A, 44,891 for Instance B).