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**

   A greedy approach where new nodes are *only* inserted at the end of the current sequence.

```
1. Start the tour with Tour = [StartNode].
2. Repeat until Size(Tour) = K:
    ○ Identify the current edge closing the cycle: (LastNode -> FirstNode).
    ○ Search all unused nodes (CandNode).
    ○ Select the CandNode that minimizes the total objective increase
      (Delta) when replacing the closing edge:
      $Delta = D[LastNode][CandNode] + D[CandNode][FirstNode] + C[CandNode]$
    ○ Add the selected CandNode to the end of the tour.
```

3. **Nearest Neighbour all-positions**

   A two-stage greedy approach: first selecting the best candidate, then finding its best position.

```
1. Start the tour with Tour = [StartNode].
2. Select the second node that minimizes $D[StartNode][SecondNode] + C[SecondNode]$.
3. Repeat until Size(Tour) = K:
    a. Selection Stage: Find the unused node (NextNode) that is "closest" to
       *any* existing node in the tour, minimizing $D[ExistingNode][NextNode] + C[NextNode]$.
    b. Insertion Stage: Find the edge (A -> B) in the current cycle where
       inserting NextNode minimizes the distance increase only:
       $Increase = D[A][NextNode] + D[NextNode][B] - D[A][B]$
    c. Insert NextNode into the tour at that best position.
```

4. **Greedy Cycle**

   A single-stage greedy approach that simultaneously finds the best candidate node and its best insertion position.

   1. Start the tour with Tour = [StartNode].
   2. Select the second node that minimizes 2 * D[StartNode][SecondNode] + C[SecondNode].
   3. Repeat until $Size(Tour) = K:
      - Search over all unused nodes (CandNode) and all existing edges (A -> B).
      - Identify the single pair (CandNode, edge A -> B) that minimizes the total objective increase Delta:
        Delta = D[A][CandNode] + D[CandNode][B] - D[A][B] + C[CandNode]
      - Insert that CandNode into the tour at the position of the corresponding edge.

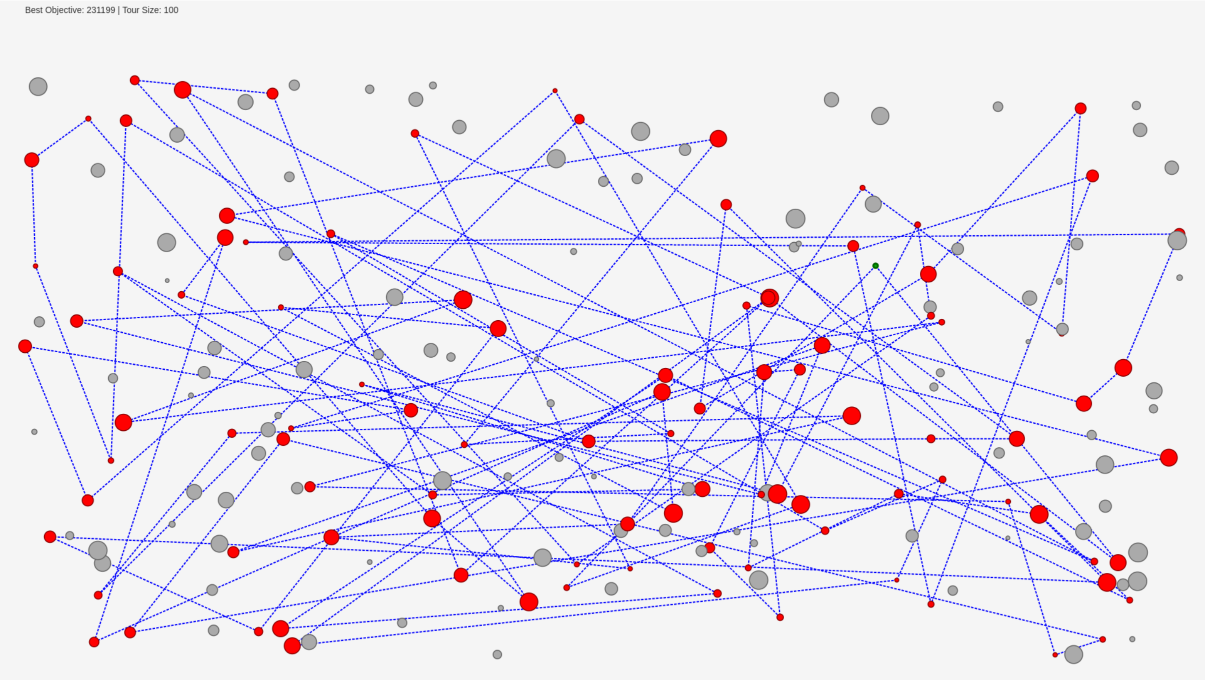
Results and Analysis

Algorithm	**Instance A**			**Instance B**		
	Min	**Max**	**Avg**	**Min**	**Max**	**Avg**
Random	240576	297408	264627	187492	239713	212299
NN end-only	89198	120393	104013	62606	77453	69764.4
NN all-pos	71515	73823	72343.6	47295	51030	48989.3
Greedy Cycle	71488	74410	72636	49001	57324	51400.6

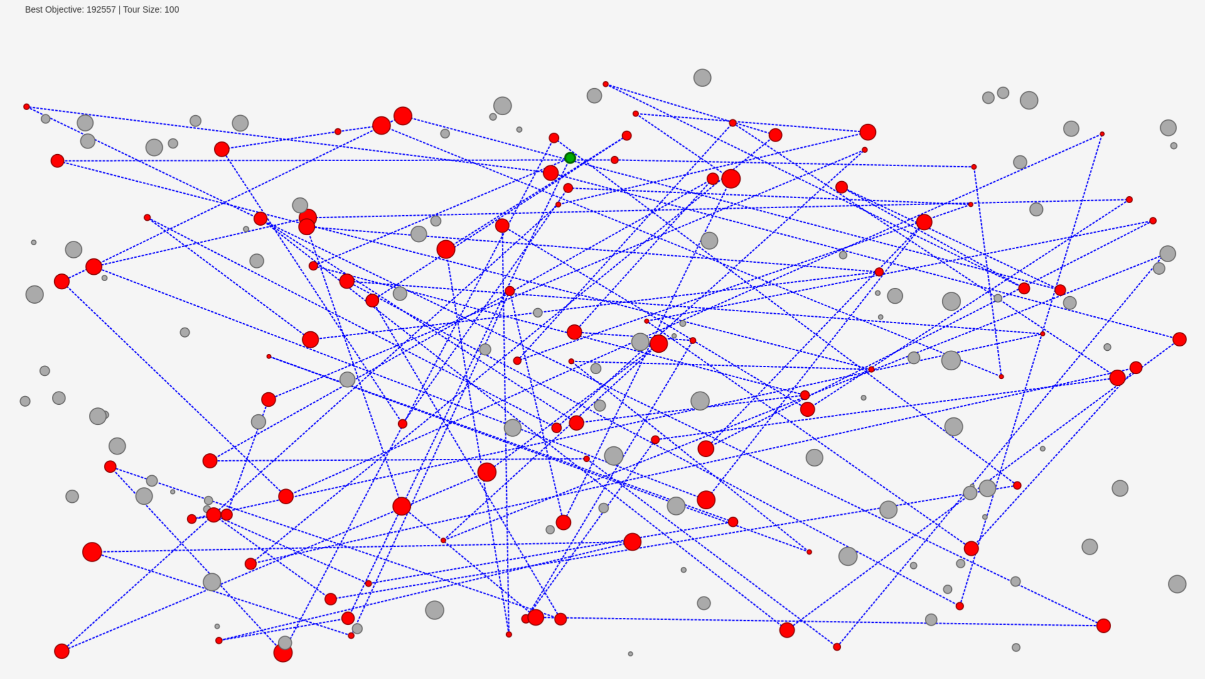
Visual Comparisons (Visual Comparision)

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.

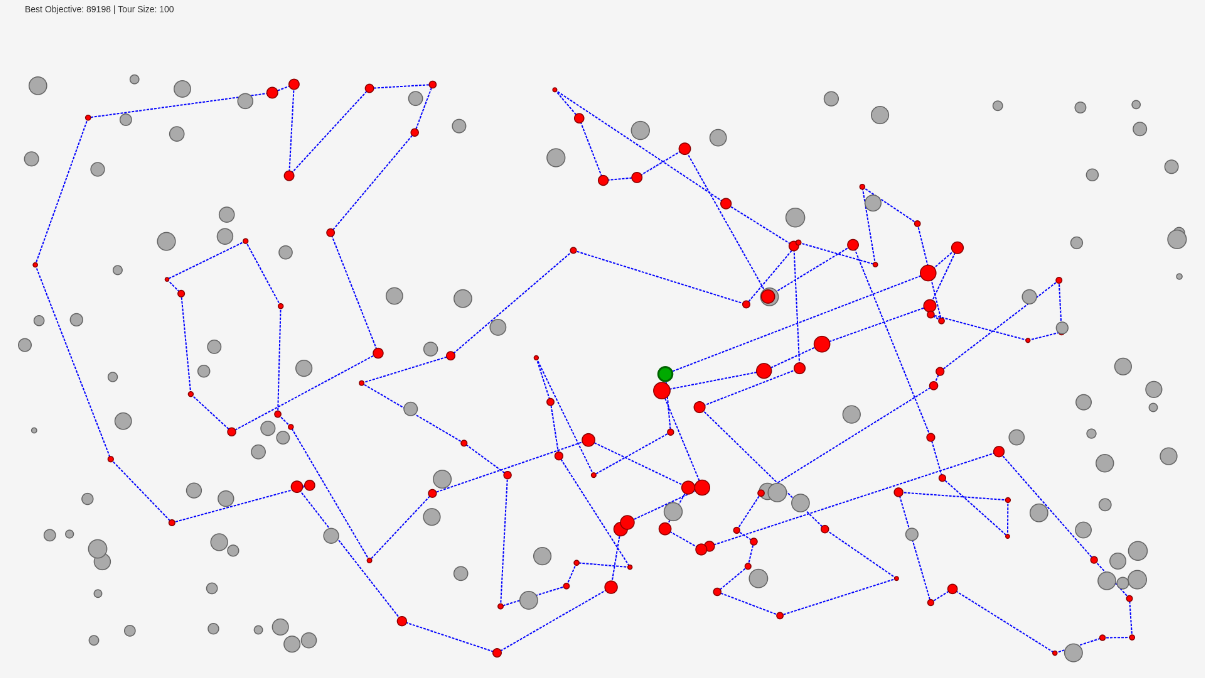
- Random algorithm
  - Instance A:



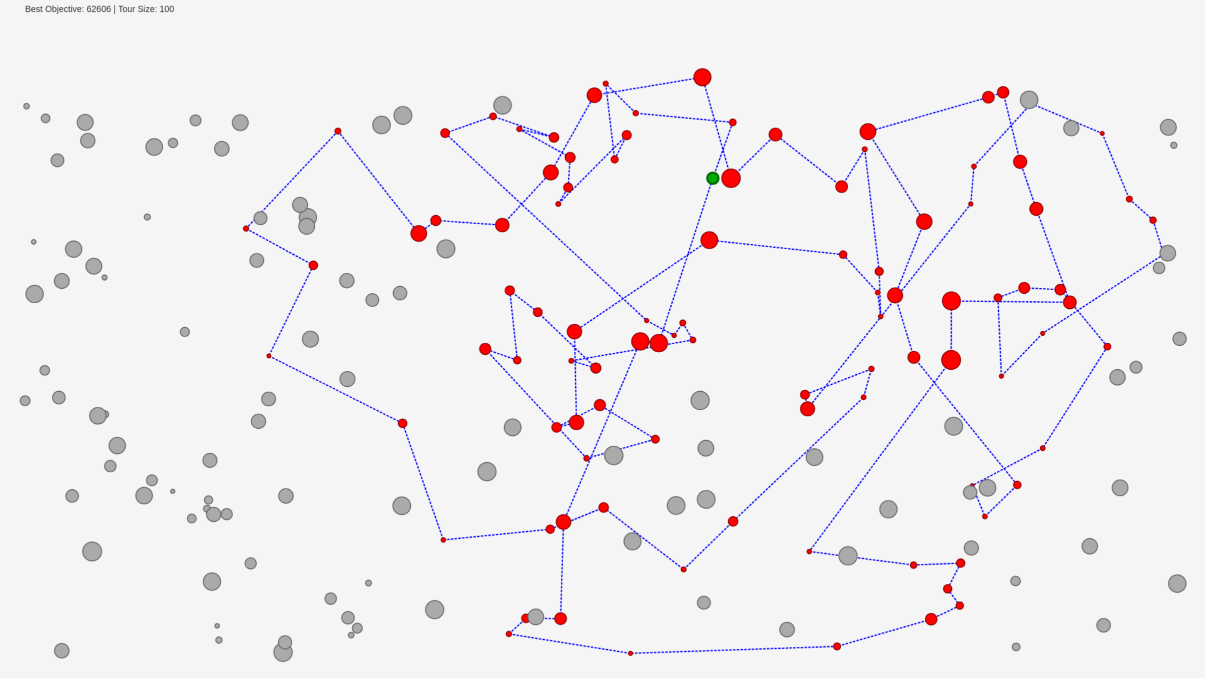
  - Instance B:



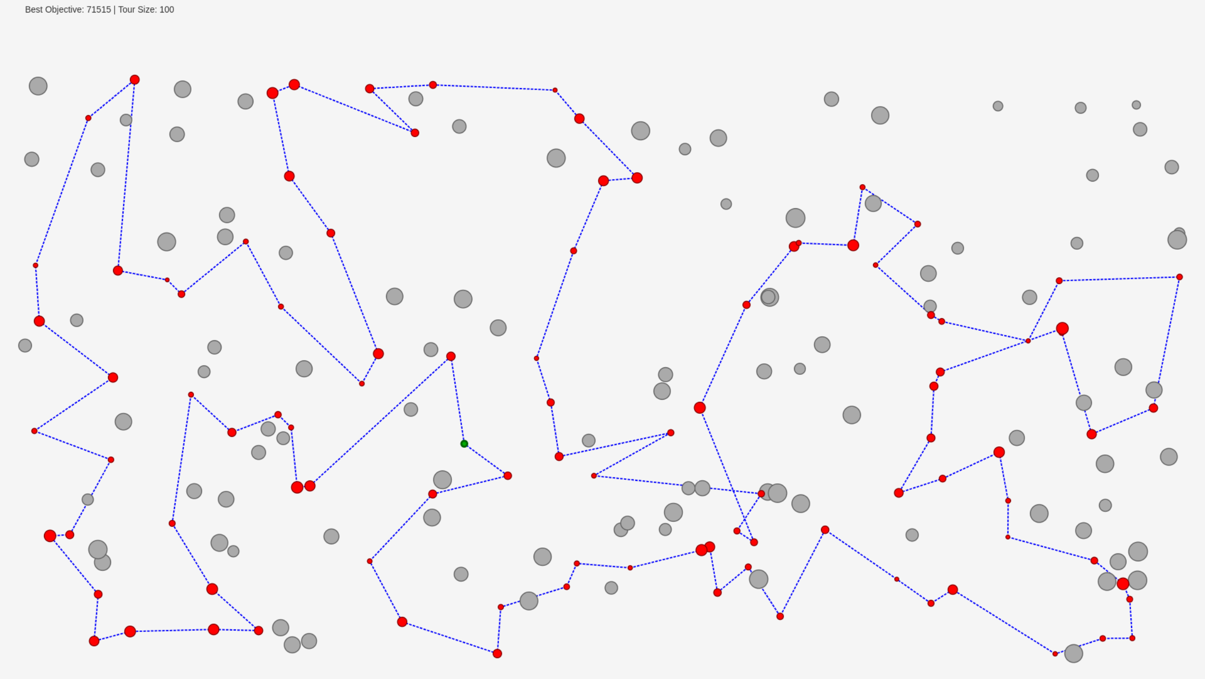
- Nearest Neighbour end-only
  - Instance A:



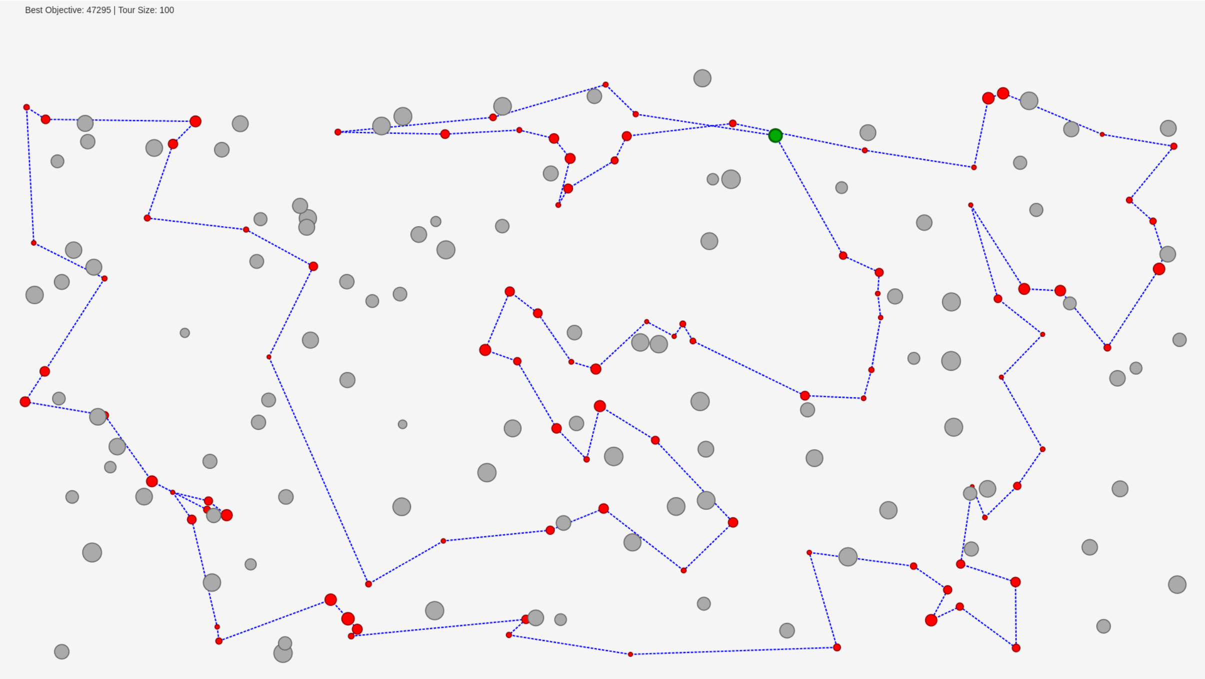
  - Instance B:



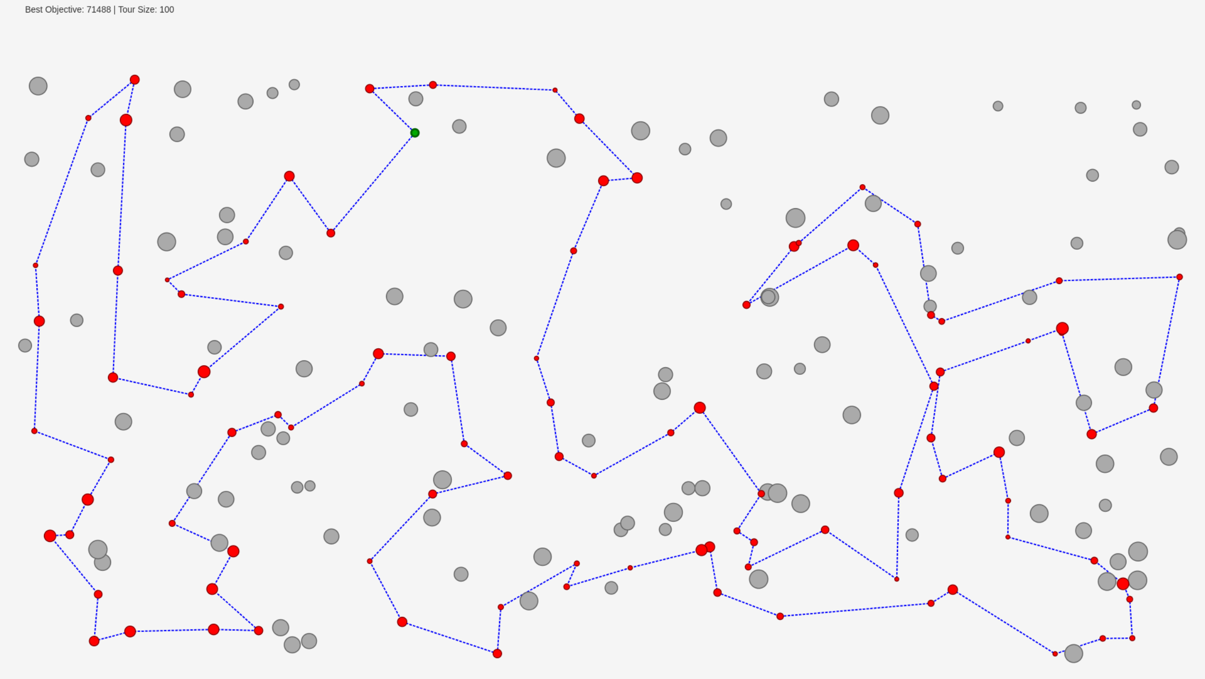
- Nearest Neighbour all-positions
    - Instance A:



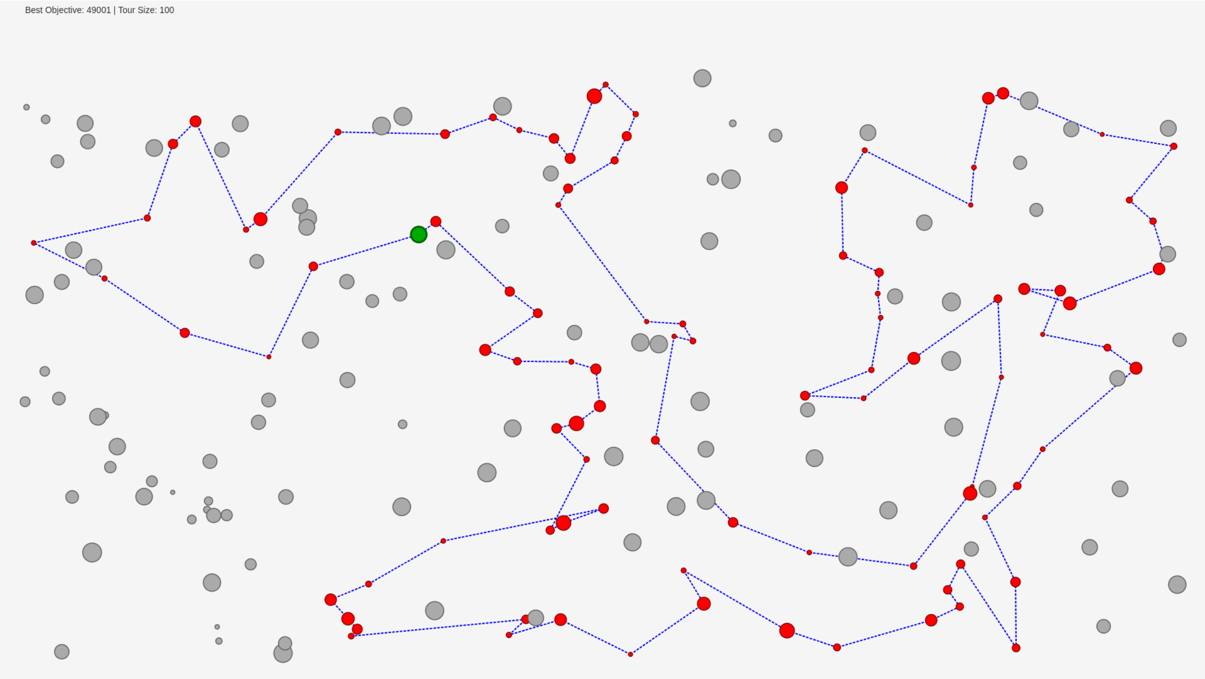
    - Instance B:



- Greedy Cycle
  - Instance A:



  - Instance B:



Best Solutions

- Random algorithm
    - Instance A: 172,17,124,186,0,30,48,182,52,80,15,133,167,91,45,165,81,113,154,161,157,74,13,110,9,160,42,86,2,89,24,60,37,92,82,56,115,131,69,146,116,54,39,140,40,53,105,104,101,4,46,126,118,137,178,8,136,199,73,61,181,149,11,16,193,135,84,32,132,188,123,152,38,109,174,107,90,7,66,112,10,173,36,134,190,5,103,63,44,97,49,79,26,158,170,168,98,145,119,128
    - Instance B: 13,51,143,56,191,146,57,126,40,75,79,123,120,55,116,122,104,152,112,73,193,185,98,127,20,173,102,150,99,170,130,70,184,136,172,26,29,47,41,174,14,32,67,148,159,43,36,2,15,118,91,156,46,132,194,45,164,5,81,121,153,33,124,100,145,140,86,192,147,178,25,195,183,138,155,52,111,61,35,49,189,151,8,187,94,177,12,171,108,168,53,3,167,87,139,175,196,58,80,142

- Nearest Neighbour end-only
    - Instance A: 12,94,63,176,80,79,53,180,154,135,133,151,59,51,137,148,62,49,144,14,106,178,185,40,165,52,55,152,97,1,101,86,75,120,2,124,167,9,15,183,89,23,186,114,37,102,57,92,78,145,129,44,25,16,171,175,113,31,179,26,100,121,189,122,162,123,65,116,115,139,193,41,42,43,118,46,0,143,117,68,93,140,18,22,160,184,149,131,127,70,158,136,182,19,130,111,128,3,138,32
    - Instance B: 189,155,3,70,145,15,168,195,13,169,132,188,6,29,0,109,35,33,160,11,139,138,182,8,111,144,104,56,49,69,34,18,62,55,152,170,184,167,84,161,126,43,134,85,147,90,51,121,25,177,21,82,77,81,106,124,143,159,183,140,28,20,148,47,94,185,86,95,130,99,179,166,176,113,194,128,83,174,53,4,149,199,9,22,181,110,153,163,103,89,127,165,187,141,36,61,91,87,39,12

- Nearest Neighbour all-positions
    - Instance A: 151,51,149,131,65,116,43,42,184,84,112,4,190,10,177,30,54,160,34,181,146,22,18,108,159,193,41,139,115,59,118,46,68,140,93,0,117,143,183,89,186,23,137,176,80,79,94,63,152,97,1,124,148,9,62,102,144,14,49,178,106,185,165,90,81,196,40,119,52,55,57,129,92,179,145,78,31,56,113,175,171,16,25,44,120,2,75,101,86,26,100,53,180,154,135,70,127,123,162,133
    - Instance B: 184,34,55,18,62,124,106,143,35,109,0,29,160,33,11,139,182,138,104,8,144,111,81,77,82,21,177,5,121,51,90,122,133,10,107,40,63,135,38,27,1,198,31,73,54,117,193,190,80,45,142,78,175,61,36,141,187,153,163,89,165,127,137,114,103,176,113,194,166,86,185,95,183,130,99,179,66,94,47,148,60,20,28,149,4,140,152,155,15,145,195,168,13,132,169,6,147,188,70,3

- Greedy Cycle
    - Instance A: 0,46,68,139,193,41,115,5,42,181,159,69,108,18,22,146,34,160,48,54,30,177,10,190,4,112,84,35,184,43,116,65,59,118,51,151,133,162,123,127,70,135,180,154,53,100,26,86,75,44,25,16,171,175,113,56,31,78,145,179,92,57,52,185,119,40,196,81,90,165,106,178,14,144,62,9,148,102,49,55,129,120,2,101,1,97,152,124,94,63,79,80,176,137,23,186,89,183,143,117
    - Instance B: 85,51,121,131,135,63,122,133,10,90,191,147,6,188,169,132,13,161,70,3,15,145,195,168,29,109,35,0,111,81,153,163,180,176,86,95,128,106,143,124,62,18,55,34,170,152,183,140,4,149,28,20,60,148,47,94,66,22,130,99,185,179,172,166,194,113,114,137,103,89,127,165,187,146,77,97,141,91,36,61,175,78,142,45,5,177,82,87,21,8,104,56,144,160,33,138,182,11,139,134

Conclusions

- The Random algorithm served as a poor baseline, yielding the highest objective function values (e.g., average of 264,627 for Instance A) due to its non-deterministic selection and lack of path optimization.
- The greedy approaches—Nearest Neighbour (NN) end-only, NN all-positions, and Greedy Cycle—significantly outperformed the random method, reducing the average objective value by over 60%.
- Algorithms that incorporated the node's potential insertion position into the tour (NN all-positions and Greedy Cycle) achieved the best results, as optimizing the path structure is crucial for minimizing the combined objective (path length + node cost).
- The Greedy Cycle and NN all-positions methods produced tours that were visibly tighter and more efficient compared to the random baseline, demonstrating a clear preference for including lower-cost nodes (smaller red dots) into the cycle.