# **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.
4. **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 $Size(Tour) = K:
   1. Selection Stage: Find the unused node ($NextNode$) that is "closest" to *any* existing node in the tour, minimizing $D[ExistingNode][NextNode] + C[NextNode]$.
   2. Insertion Stage: Find the edge $(A \to B)$ in the current cycle where inserting $NextNode$ minimizes the distance increase only:  
       $$Increase = D[A][NextNode] + D[NextNode][B] - D[A][B]$$
   3. Insert $NextNode$ into the tour at that best position.
4. **Greedy Cycle**

## 

## **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)**

* Random algorithm
* Nearest Neighbour end-only
* Nearest Neighbour all-positions
* Greedy Cycle

### **Best Solutions**

* Random algorithm
* Nearest Neighbour end-only
* Nearest Neighbour all-positions
* Greedy Cycle

## **Conclusions**

* **Conclusions**