

*Radosław Bodus 145457*

*Hubert Radom 145445*

Source code: <https://github.com/HubertRadom/EvolutionaryComputation/lab1/lab1.ipynb>

# Greedy heuristics

## Problem description

We are given three columns of integers with a row for each node. The first two columns contain x and y coordinates of the node positions in a plane. The third column contains node costs. The goal is to select exactly 50% of the nodes (if the number of nodes is odd we round the number of nodes to be selected up) and form a Hamiltonian cycle (closed path) through this set of nodes such that the sum of the total length of the path plus the total cost of the selected nodes is minimized. The distances between nodes are calculated as Euclidean distances rounded mathematically to integer values. The distance matrix should be calculated just after reading an instance and then only the distance matrix (no nodes coordinates) should be accessed by optimization methods to allow instances defined only by distance matrices.

## PSEUDOCODE

*limit* = 50% length of data

### Create cost matrix:

```
distance_matrix = []  
data = load data from csv to array with shape (n,3)  
costs = last column of data  
For i from 0 to n:  
    row = []  
    For j from 0 to n:  
        if i != j:  
            Append distance from data[i] to data[j] to row  
        else:  
            Append 0 to row  
    Append row to distance_matrix  
Add costs to distance_matrix
```

### Random solution:

```
nodes_id = [0,1,2,...,n]  
Shuffle nodes_id  
Return nodes_id from 0 to limit
```

**Nearest neighbor(*current\_node*):**

If not *current\_node* then:

*current\_node* = random node

*nodes\_left* = set of all nodes ids except *current\_node*

*solution* = [*current\_node*]

While length of *solution* is less than *limit*:

*min\_node\_cost* = infinity

*min\_node*;

    For *next\_node* in *nodes\_left*:

*next\_cost* = *cost\_matrix*[*current\_node*][*next\_node*]

        if *next\_cost* < *min\_node\_cost* then:

*min\_node\_cost* = *next\_cost*

*min\_node* = *next\_node*

    Append *min\_node* to *solution*

    Remove *min\_node* from *nodes\_left*

Return *solution*

**Greedy cycle(*current\_node*):**

If not *current\_node* then:

*current\_node* = random node

*nodes\_left* = set of all nodes ids except *current\_node*

*solution* = [*current\_node*]

Add to solution next node with minimum cost as in nearest neighbor algorithm

While length of *solution* is less than *limit*:

*min\_delta* = infinity

*min\_node*, *insert\_position*;

    From *i* from 0 to length - 1 of *solution*:

        for *next\_node* in *nodes\_left*:

*delta* = *cost\_matrix*[*solution*[*i*]][*next\_node*] +

+ *cost\_matrix*[*next\_node*][*solution*[*i*+1]] - *cost\_matrix*[*solution*[*i*]][*solution*[*i*+1]]

        if *delta* < *min\_delta* then:

*min\_delta* = *delta*

*min\_node* = *next\_node*

*insert\_position* = *i*;

    for *next\_node* in *nodes\_left*:

*delta* = *cost\_matrix*[*solution*[-1]][*next\_node*] +

*cost\_matrix*[*next\_node*][*solution*[0]] - *cost\_matrix*[*solution*[-1]][*solution*[0]]

    if *delta* < *min\_delta* then:

*min\_delta* = *delta*

*min\_node* = *next\_node*

*insert\_position* = *i*;

    Append *min\_node* to *solution*

    Remove *min\_node* from *nodes\_left*

Return *solution*

## Tests

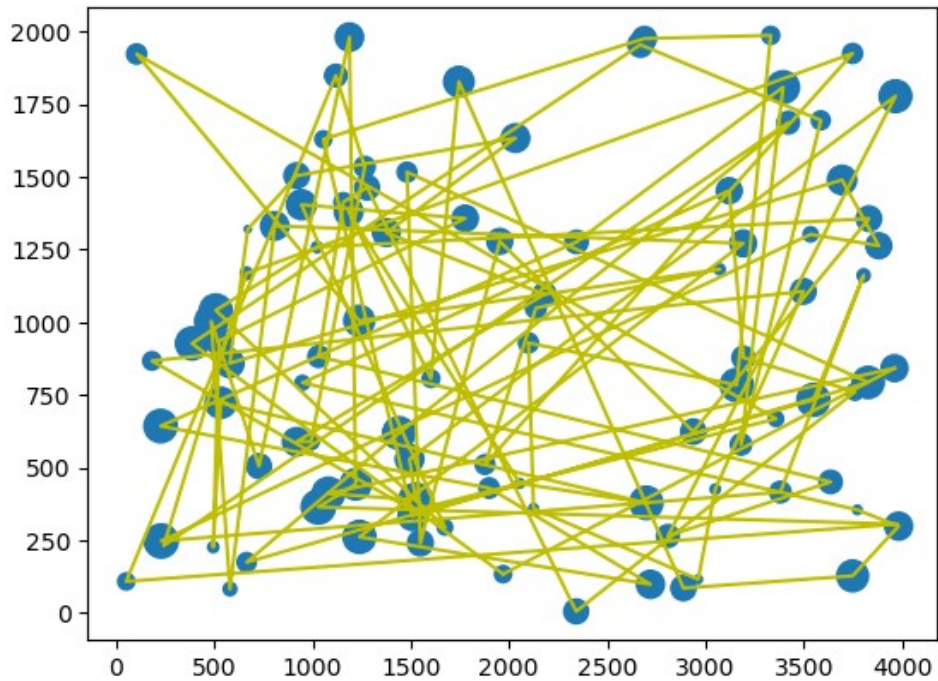
### Random Solution

Set A

AVERAGE: 264986

MAXIMUM: 293539

MINIMUM: 244780

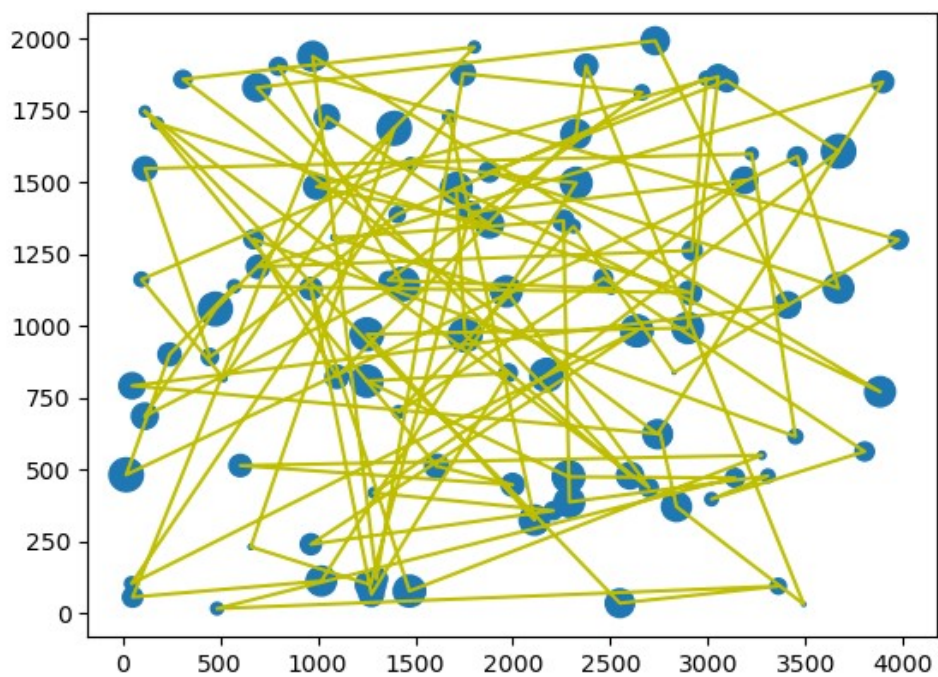


Set B

AVERAGE: 266317

MAXIMUM: 292494

MINIMUM: 240324



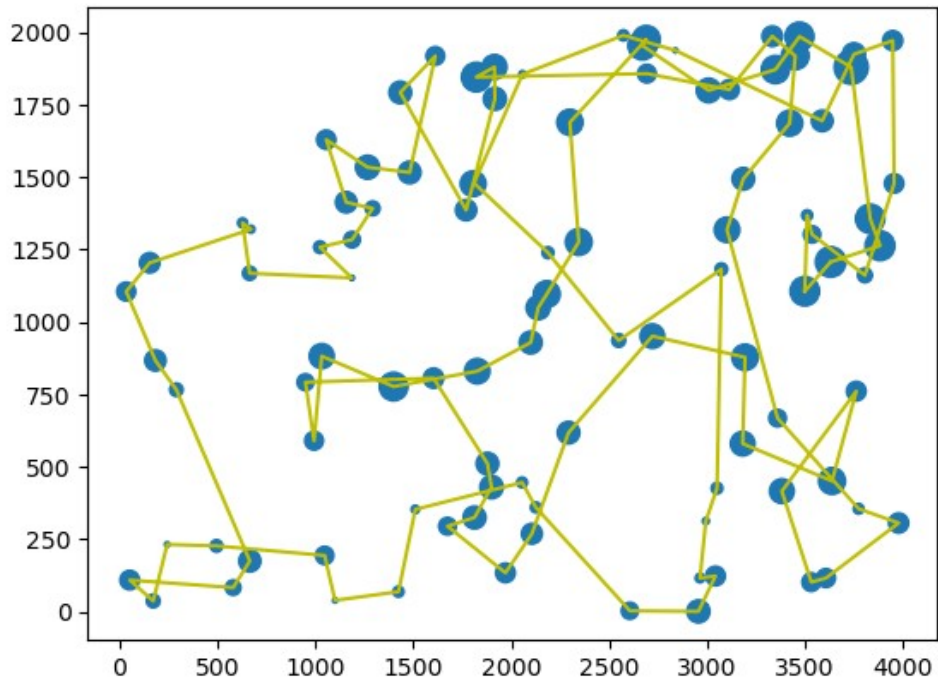
## Nearest Neighbor

Set A

AVERAGE: 87741

MAXIMUM: 95932

MINIMUM: 84840

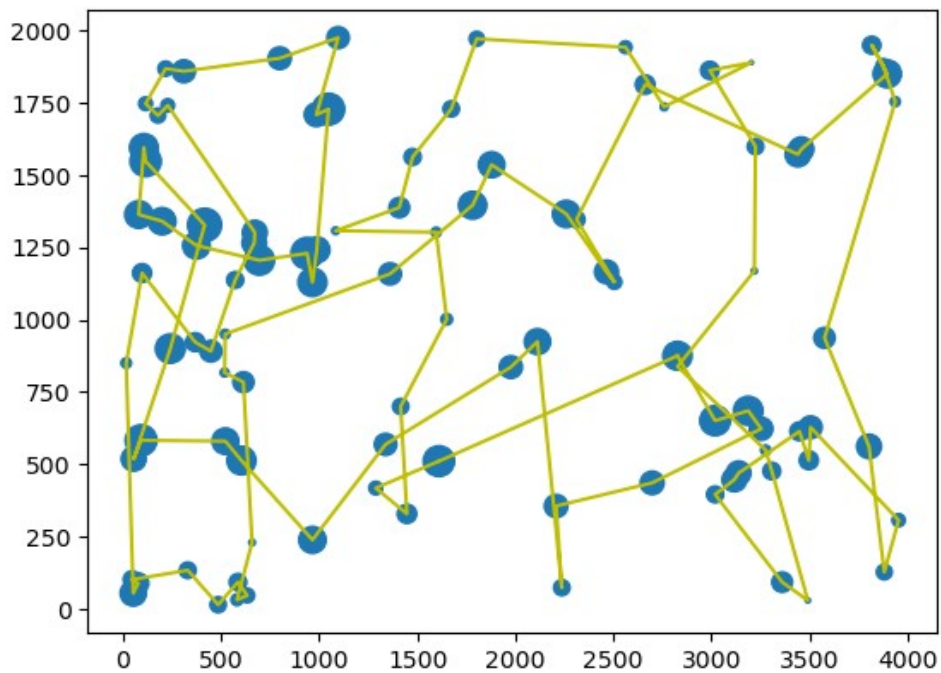


Set B

AVERAGE: 79096

MAXIMUM: 81600

MINIMUM: 77417



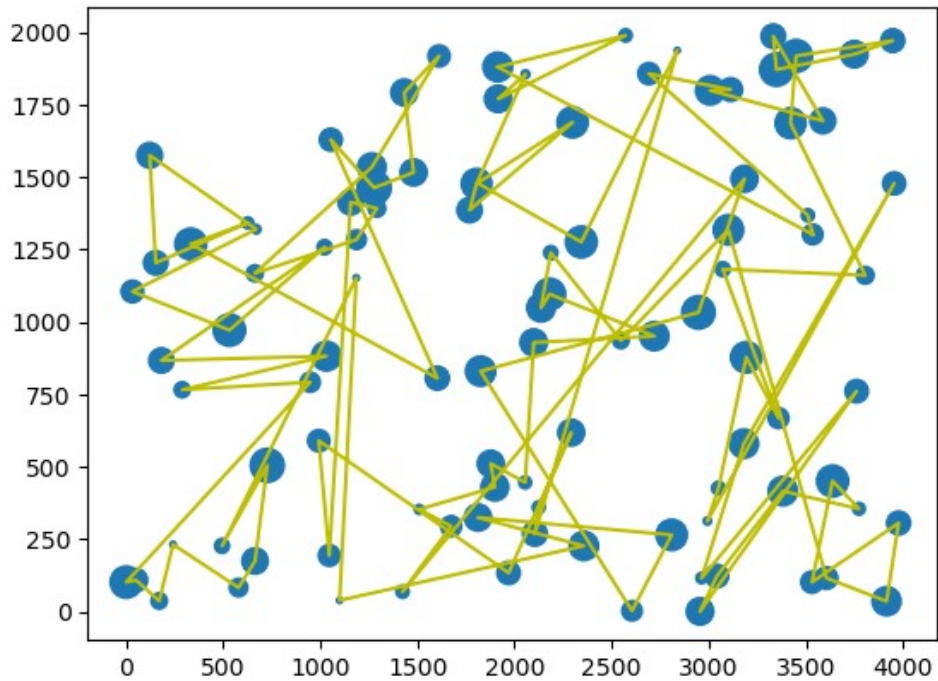
## Greedy Cycle

Set A

AVERAGE: 121599

MAXIMUM: 139172

MINIMUM: 107857

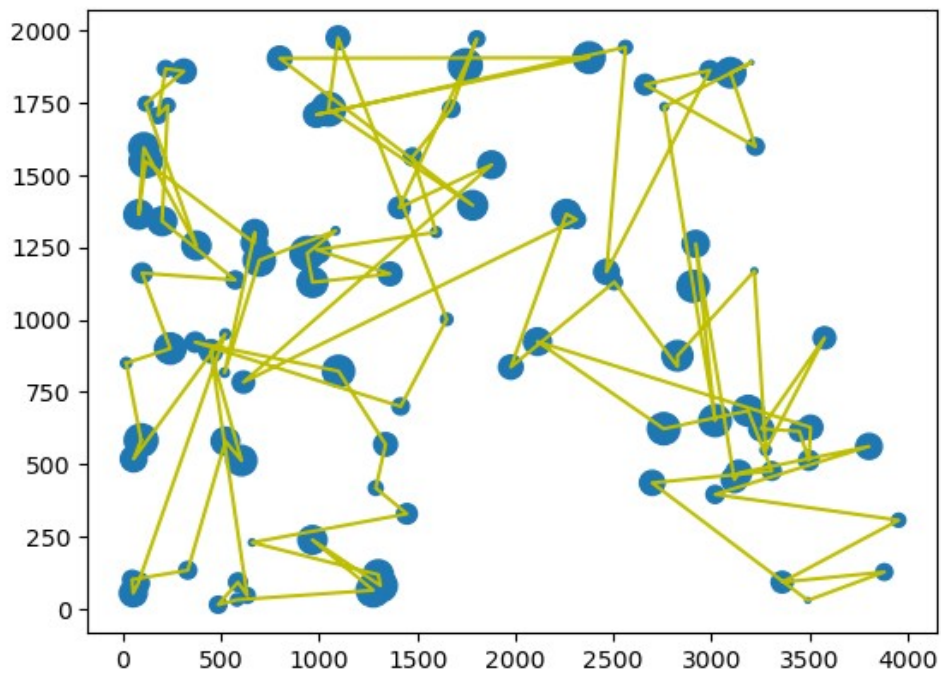


Set B

AVERAGE: 113789

MAXIMUM: 143171

MINIMUM: 97926



## Conclusion

As we expected, random solution gives very poor results and serves us only as a reference point for other algorithms. The best results are obtained by the nearest neighbor algorithm, which is also much faster than greedy cycle in terms of computational complexity, so it outclasses the competition.