Radosław Bodus 145457

Hubert Radom 145445

Source code: https://github.com/HubertRadom/EvolutionaryComputation/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% lenght 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

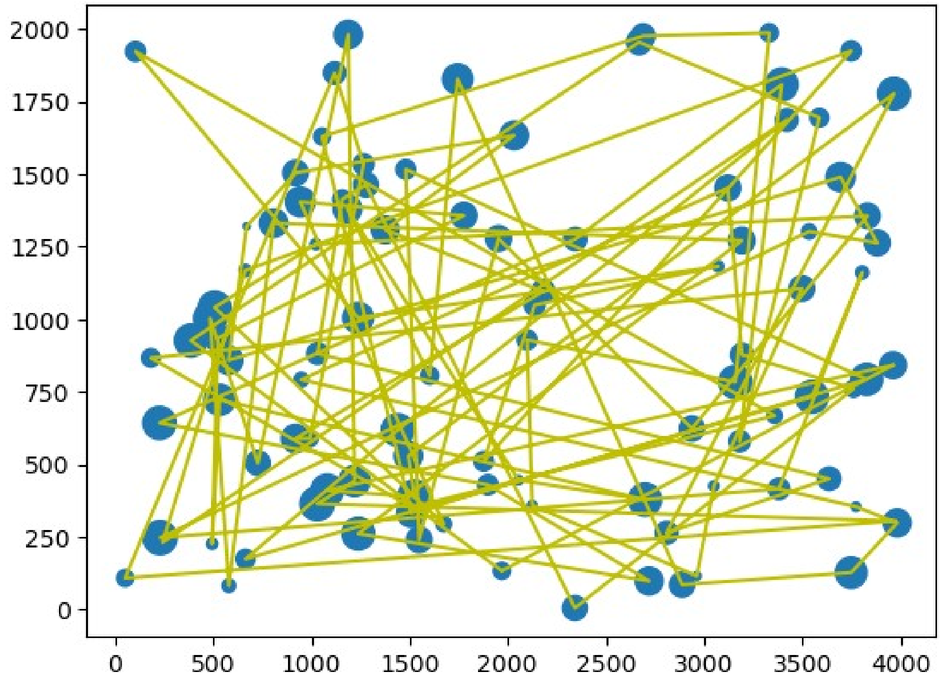
Random Solution

Set A

AVERAGE: 264986

MAXIMUM: 293539

MINIMUM: 244780

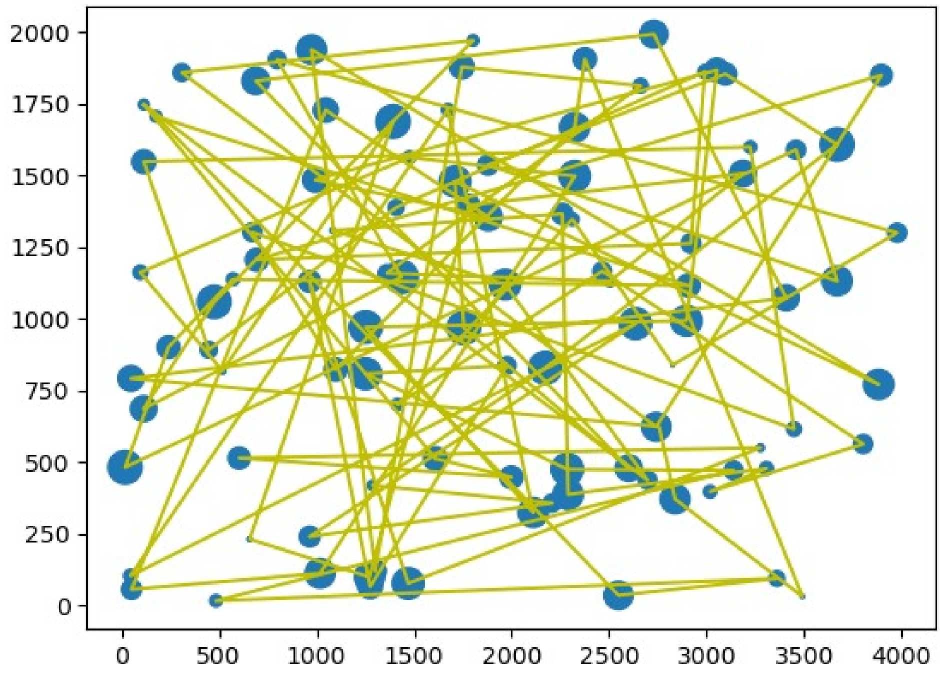


Set B

AVERAGE: 266317

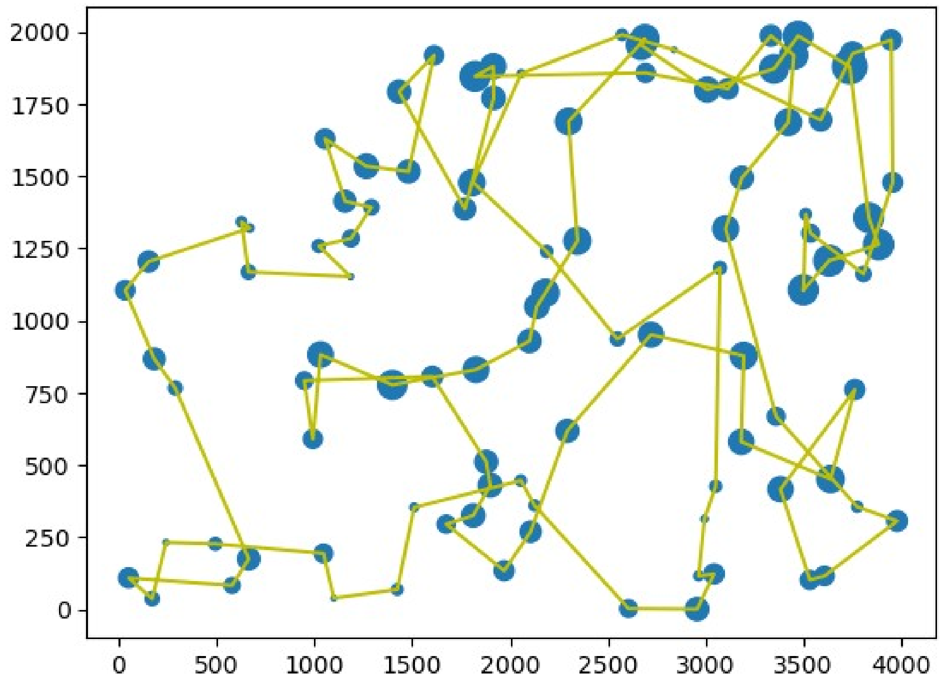
MAXIMUM: 292494

MINIMUM: 240324

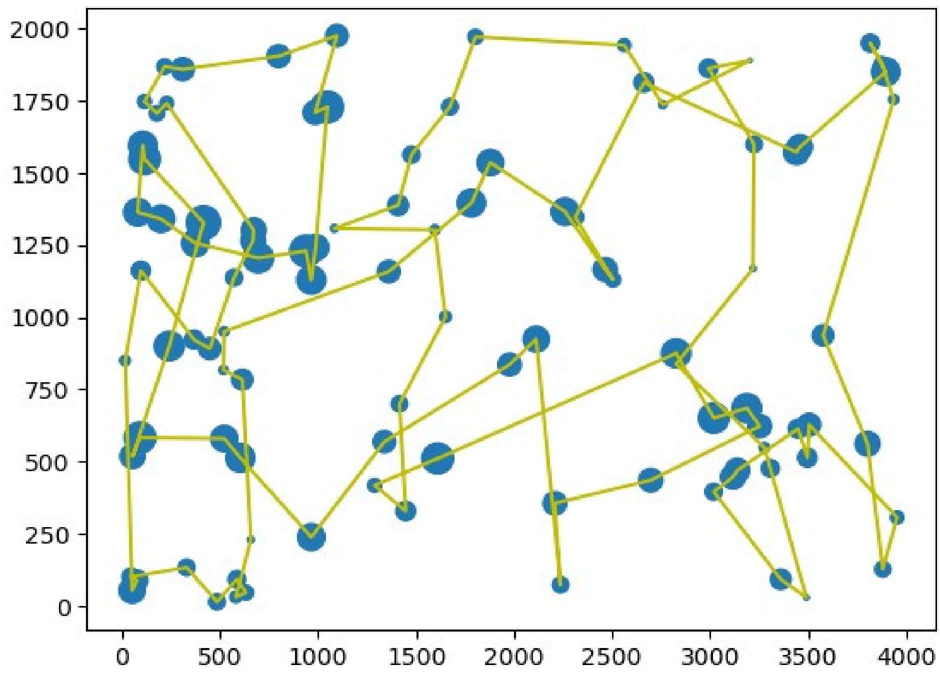


Nearest Neighbor

Set A
AVERAGE: 87741
MAXIMUM: 95932
MINIMUM: 84840

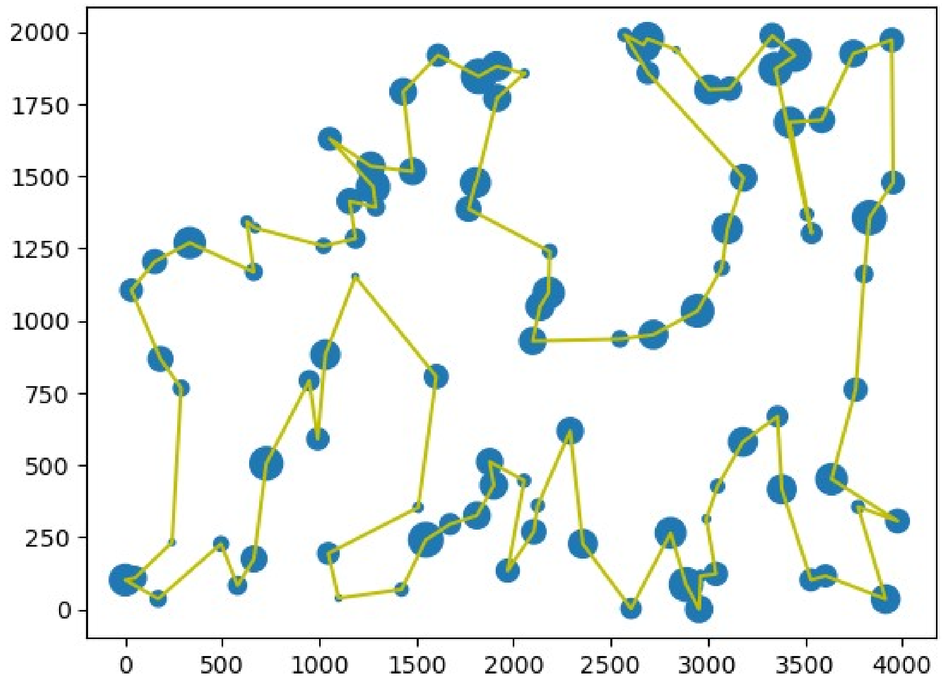


Set B
AVERAGE: 79096
MAXIMUM: 81600
MINIMUM: 77417

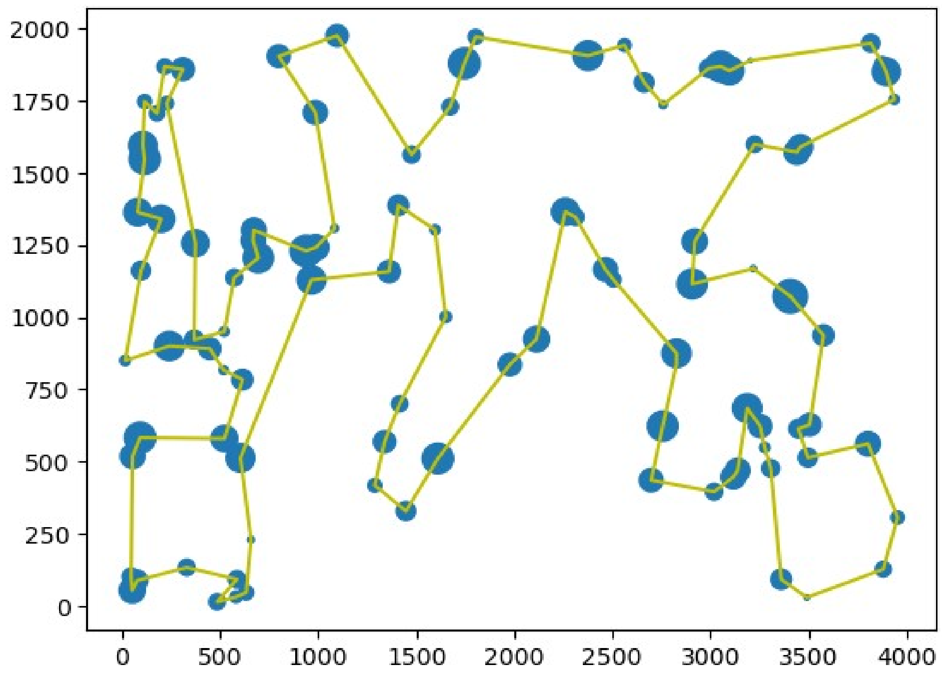


Greedy Cycle

Set A
AVERAGE: 76387
MAXIMUM: 79158
MINIMUM: 74573



Set B
AVERAGE: 69961
MAXIMUM: 75068
MINIMUM: 67684



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 greedy cycle that is much slower comparing to the nearest neighbor algorithm. In real use case we should choose between efficiency and time tradeoff.