Evolutionary Computation

Assignment 1

Source code: https://github.com/JankowskiDaniel/evolutionary-computation

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 of implemented algorithms

An additional method for calculating the total costs of selected nodes:

```
total_cost(selected_nodes, costs):
    total_cost = 0
    FOR node N in selected_nodes:
        total_cost += costs[N]
    return total_cost
```

The random solution algorithm

```
Generate_random_solution(dist_matrix, costs):
    num_nodes = dist_matrix.shape[0]
    num_select = (num_nodes + 1) // 2
    selected_nodes = randomly select num_select nodes that will be in the solution and suffle them
    total_distance = 0
    FOR N in range(1, len(selected_nodes)):
        previous_node = selected_nodes[N-1]
        current_node = selected_nodes[N]
        total_distance += dist_matrix[current_node][previous_node]
    // add the distance between the last and the first node
    total_distance += dist_matrix[selected_nodes[-1], selected_nodes[0]]
    total_nodes_cost = total_cost(selected_nodes, costs)
    return selected_nodes, total_distance+total_nodes_cost
```

The nearest neighbor algorithm

```
Generate_nearest_neighbor_solution(dist_matrix, costs, start_node):
        num_nodes = dist_matrix.shape[0]
        num_select = (num_nodes + 1) // 2
        selected_nodes = [start_node]
        unselected_nodes = {num_nodes} \ {start_node}
        total_distance = 0
        WHILE len(selected_nodes) < num_select:
                last_node = selected_nodes[-1]
                nearest_node = min(unselected_nodes,
                                key=lambda node: dist_matrix[last_node][node])
                ADD nearest_node to selected_nodes
                REMOVE nearest_node from unselected_nodes
                total_distance += dist_matrix[last_node][nearest_node]
        // add the distance between the last and the first node
        total_distance +=  dist_matrix[selected_nodes[-1], selected_nodes[0]]
        total_nodes_cost = total_cost(selected_nodes, costs)
        return selected_nodes, total_distance+total_nodes_cost
```

The greedy cycle algorithm

```
Generate_greedy_cycle_solution(dist_matrix, costs, start_node):
        num_nodes = dist_matrix.shape[0]
        num_select = (num_nodes + 1) // 2
        selected_nodes = [start_node]
        unselected_nodes = {num_nodes} \ {start_node}
        total_distance = 0
        WHILE len(selected_nodes) < num_select:
                last_node = selected_nodes[-1]
                nearest_node = min(unselected_nodes,
                                key=lambda        node:        dist_matrix[last_node][node]        +
                                dist_matrix[start_node][node])
                ADD nearest_node to selected_nodes
                REMOVE nearest_node from unselected_nodes
                total_distance += dist_matrix[last_node][nearest_node]
        // add the distance between the last and the first node
        total_distance +=  dist_matrix[selected_nodes[-1], selected_nodes[0]]
        total_nodes_cost = total_cost(selected_nodes, costs)
        return selected_nodes, total_distance+total_nodes_cost
```

Results

The random solution

	Min	**Max**	**Mean**
Instance A	245,156	290,117	266,413.17
Instance B	240,788	288,345	267,593.63
Instance C	190,445	240,929	216,693.22
Instance D	196,043	243,798	220,542.18

The nearest neighbor algorithm

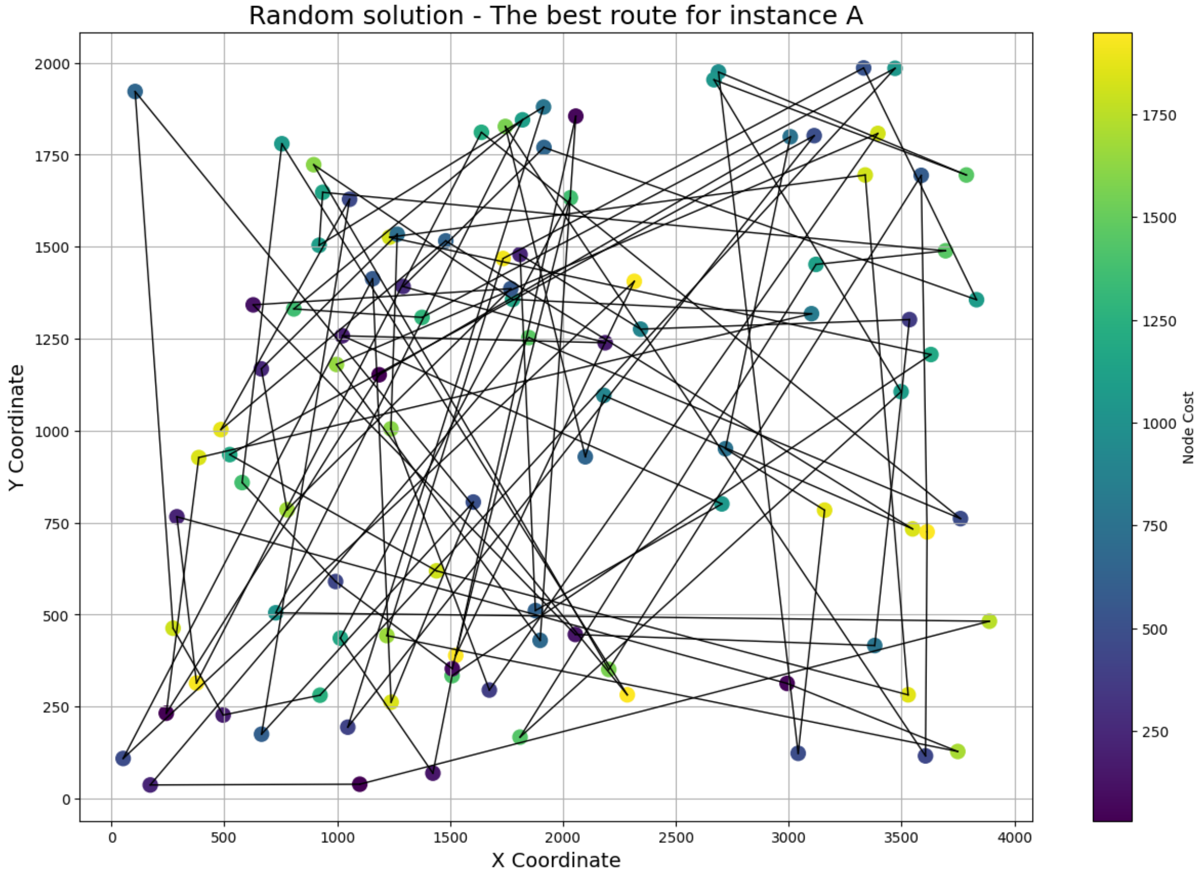
	Min	**Max**	**Mean**
Instance A	110,035	125,805	116,516.55
Instance B	109,047	124,759	116,413.93
Instance C	62,629	71,814	66,329.95
Instance D	62,967	71,396	67,119.2

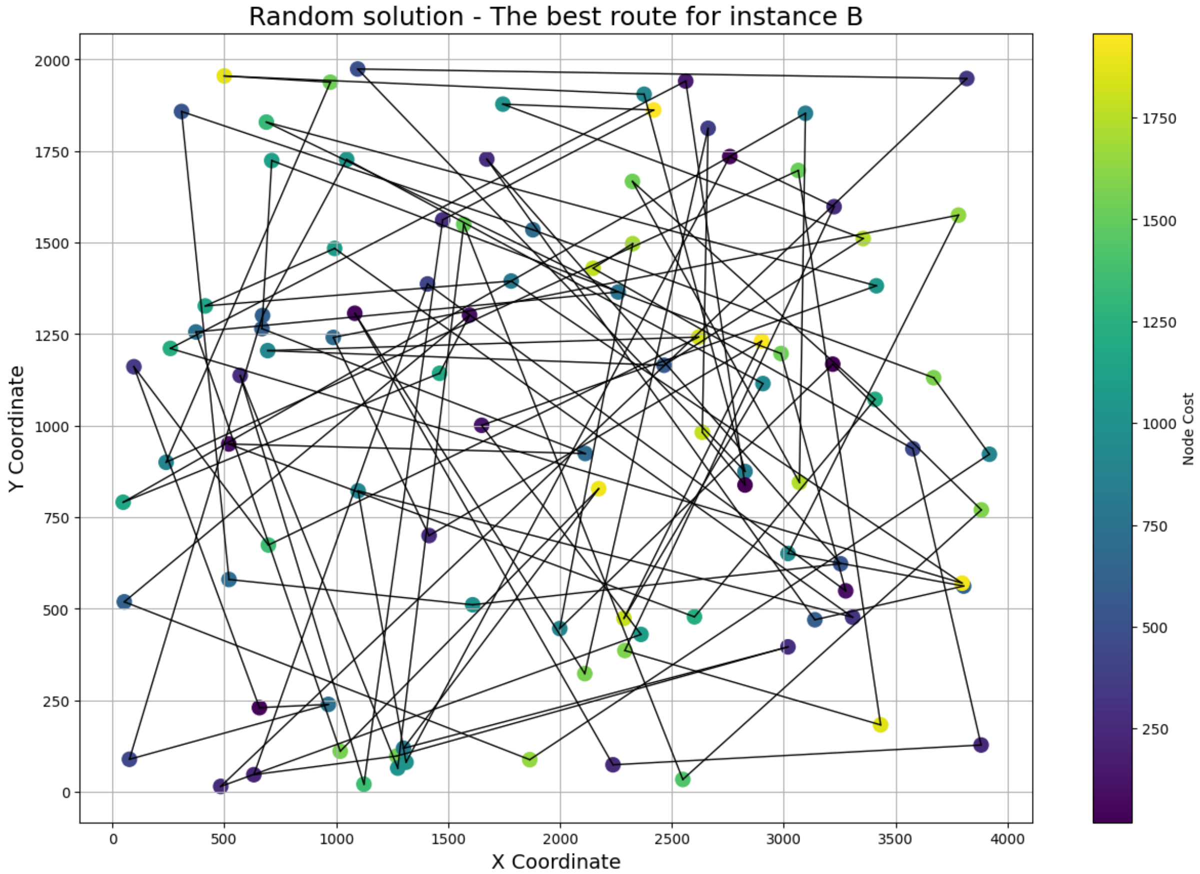
The greedy cycle algorithm

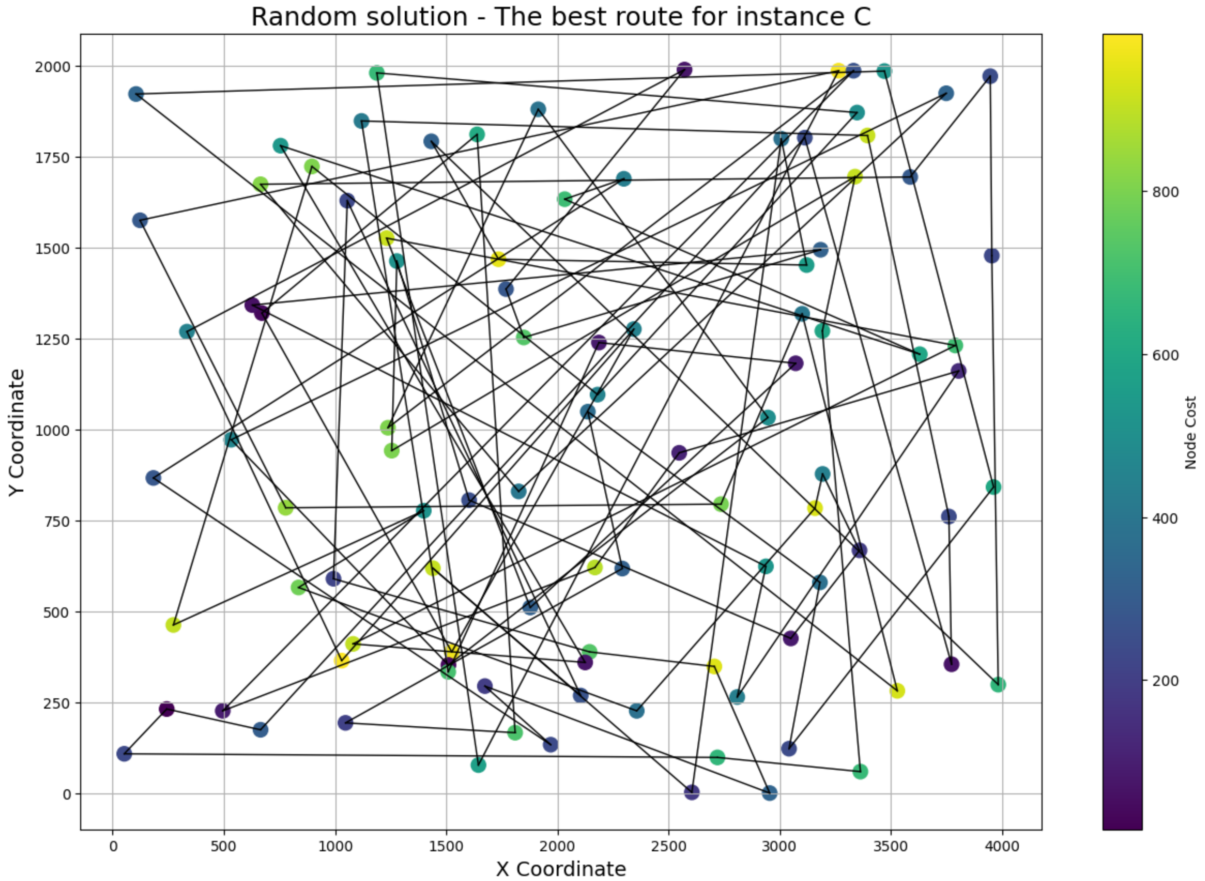
	Min	Max	Mean
Instance A	111,129	125,930	120,306.18
Instance B	108,501	129,321	119,611.18
Instance C	65,218	73,928	70,146.16
Instance D	63,694	74,892	69,988.40

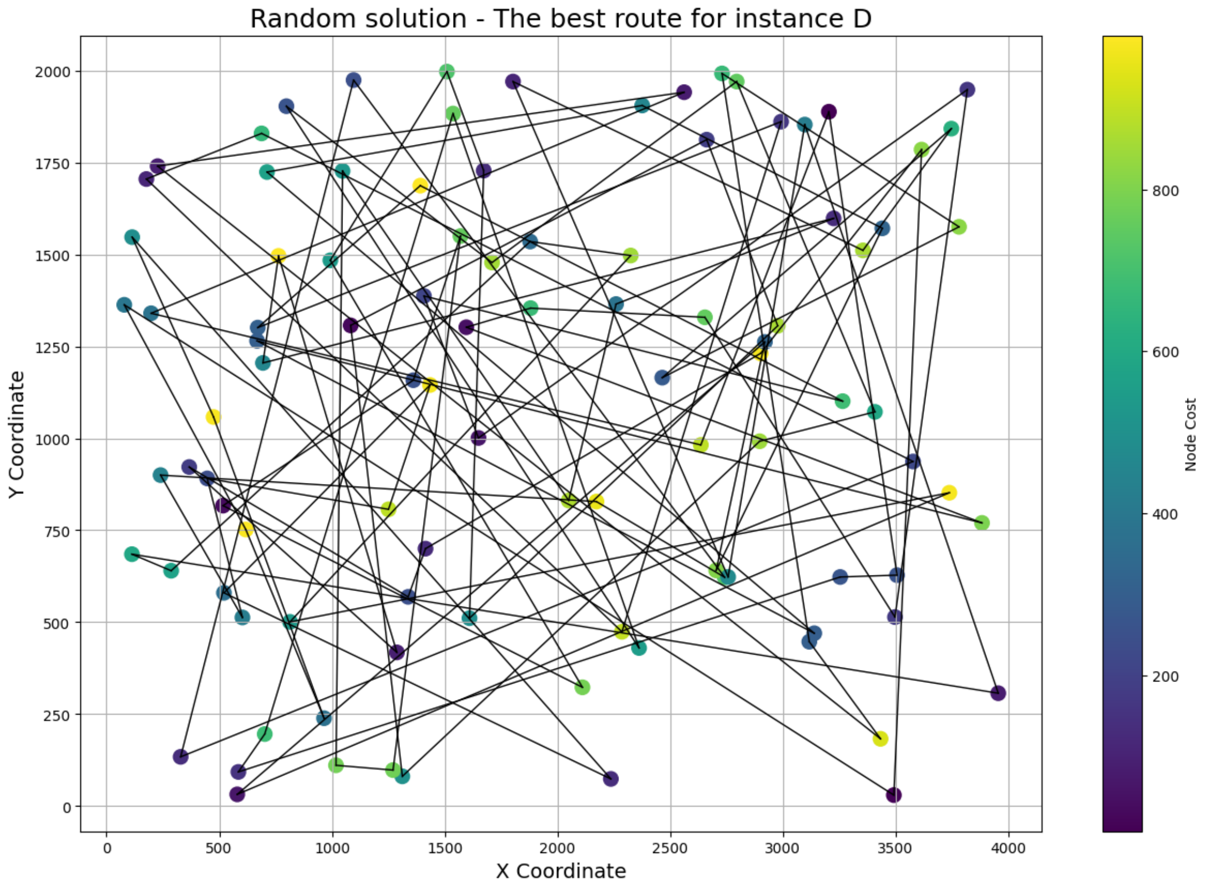
Visualizations

The random solution

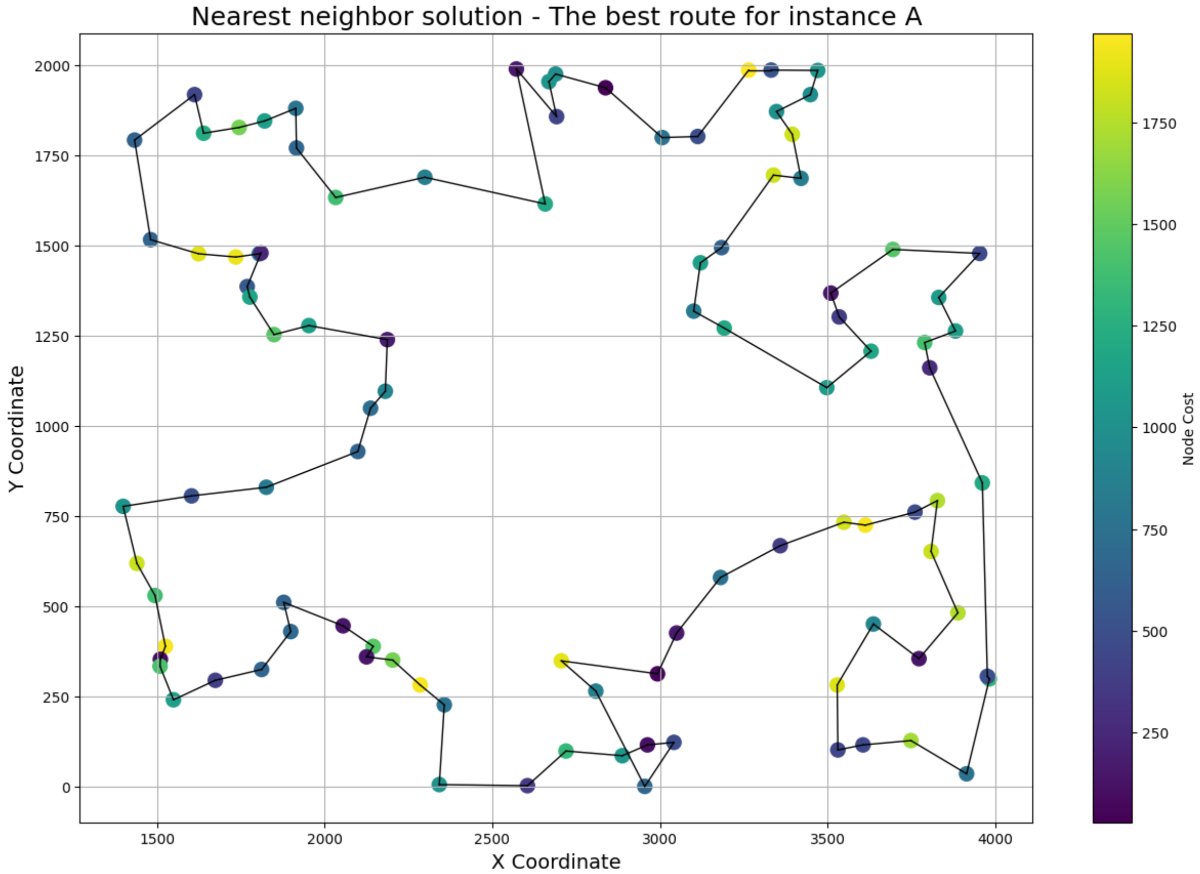


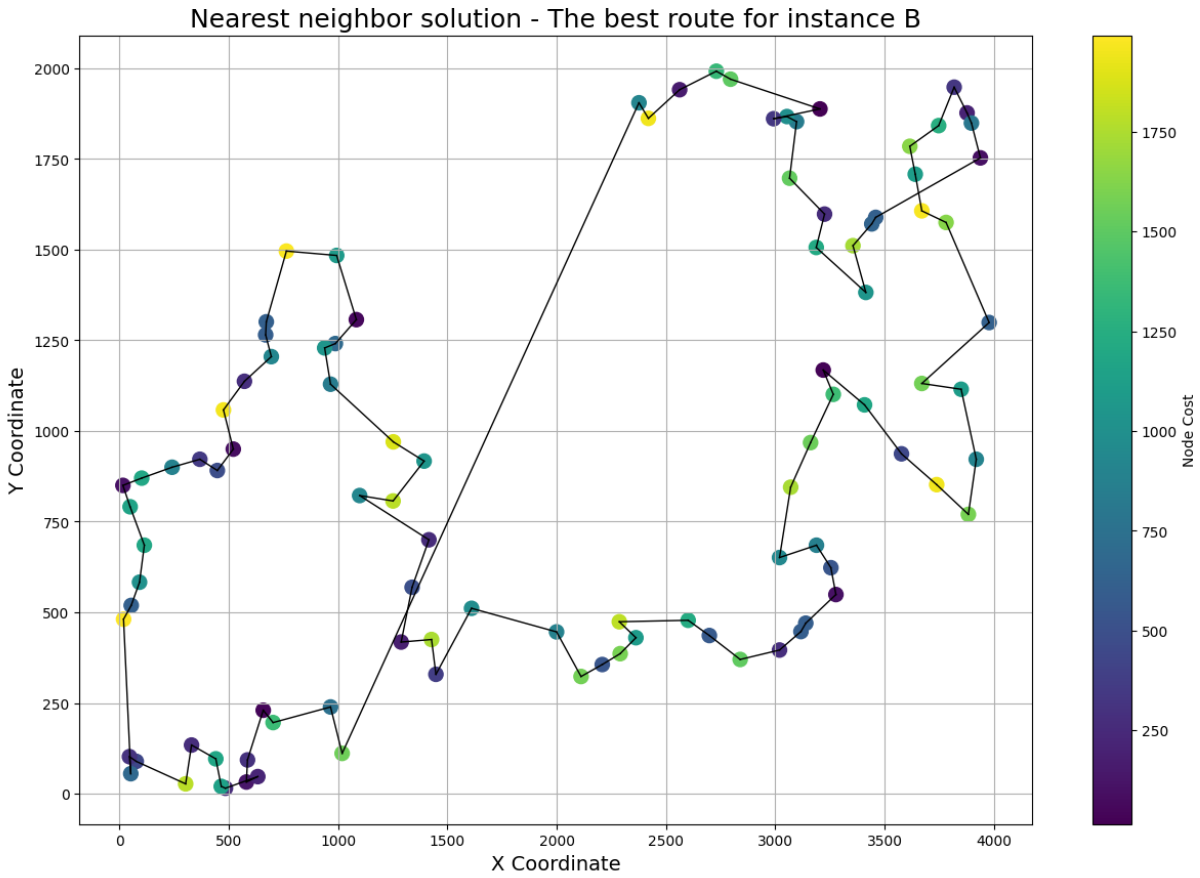



Random solution - The best route for instance C
Y Coordinate
X Coordinate
Node Cost
0
250
500
750
1000
1250
1500
1750
2000
500
1500
2000
2500
3000
3500
4000
200
400
600
800

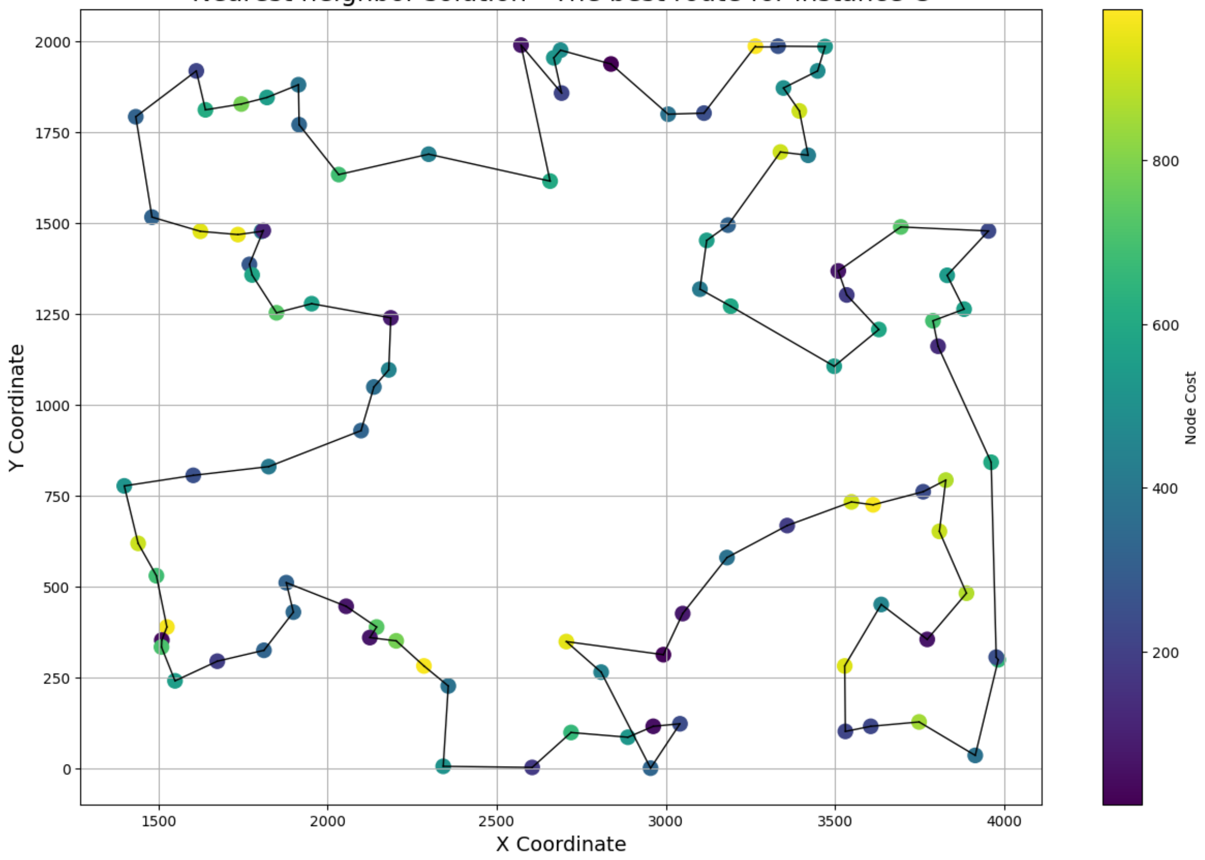

Random solution - The best route for instance D
Y Coordinate
X Coordinate
Node Cost
0
250
500
750
1000
1250
1500
1750
2000
500
1500
2000
2500
3000
3500
4000
200
400
600
800

The nearest neighbor algorithm

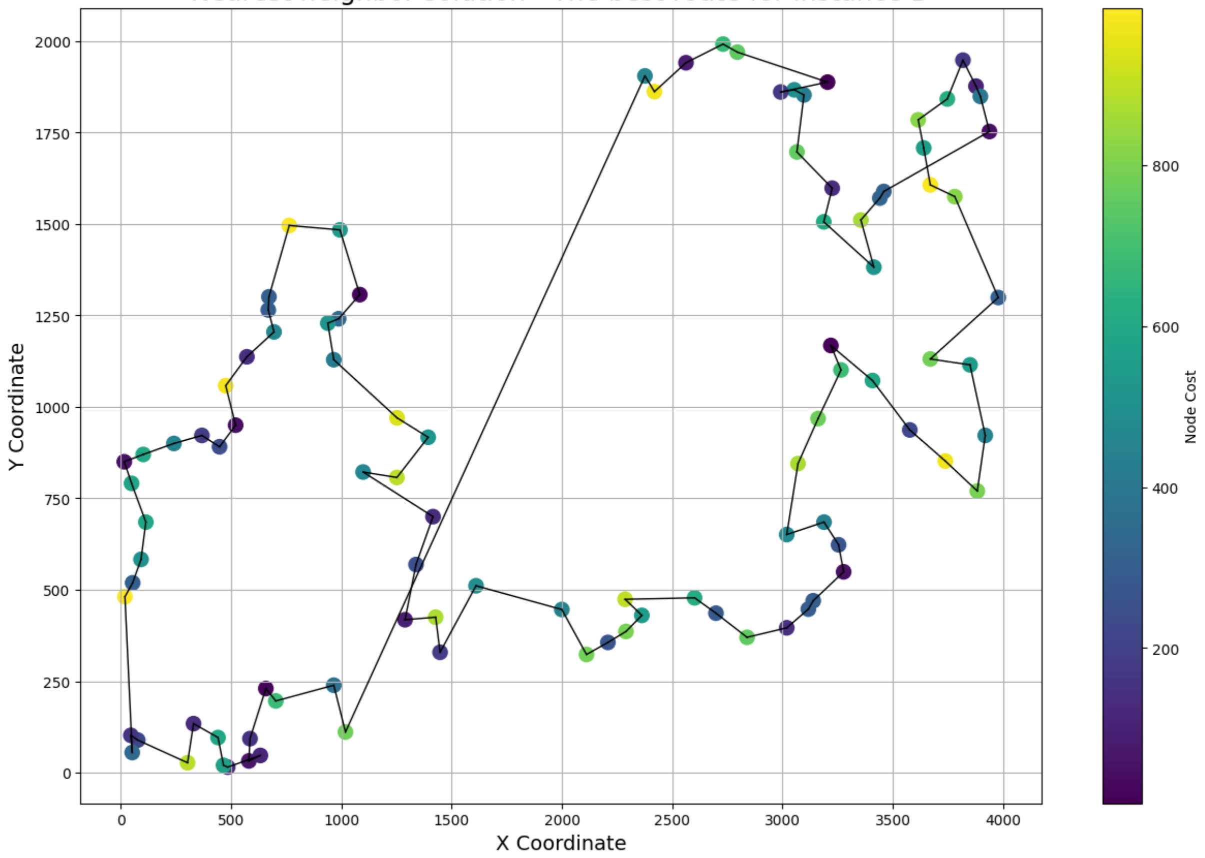




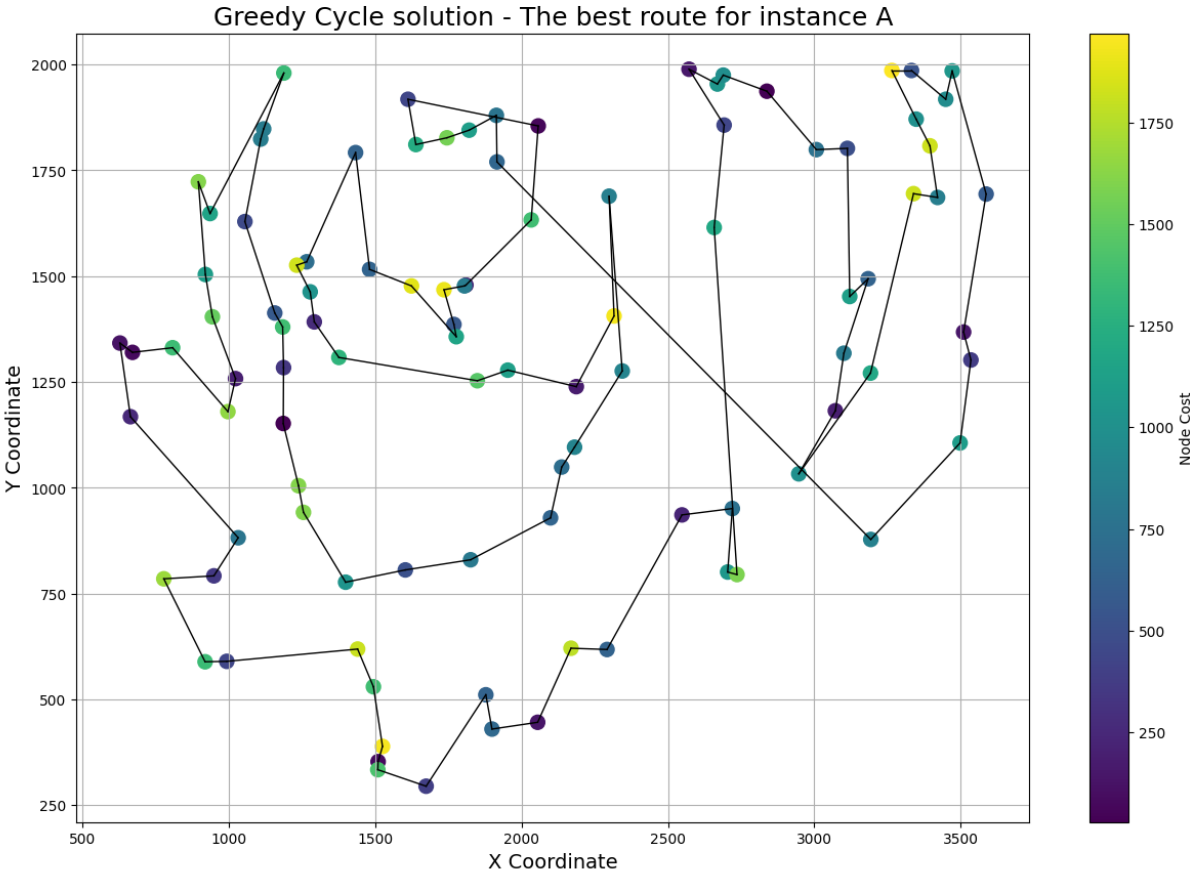
Nearest neighbor solution - The best route for instance C

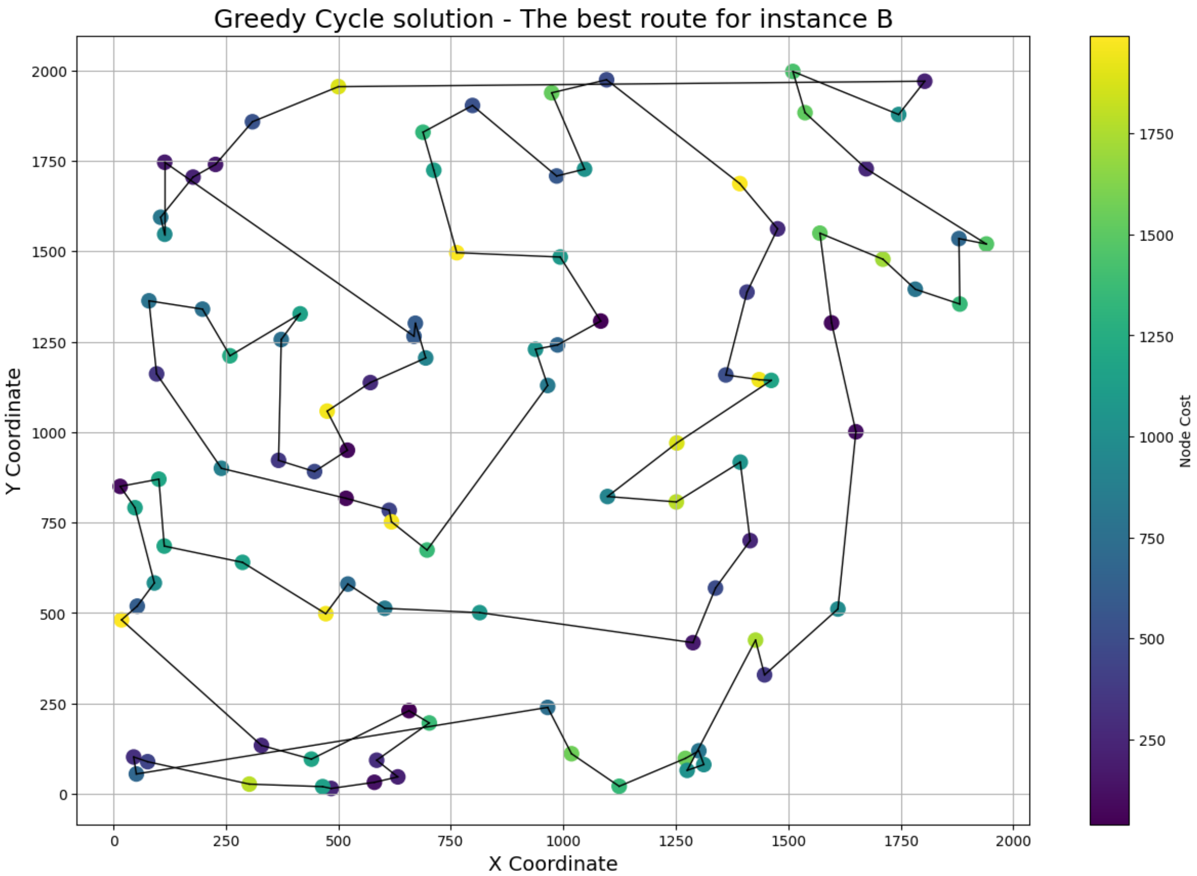


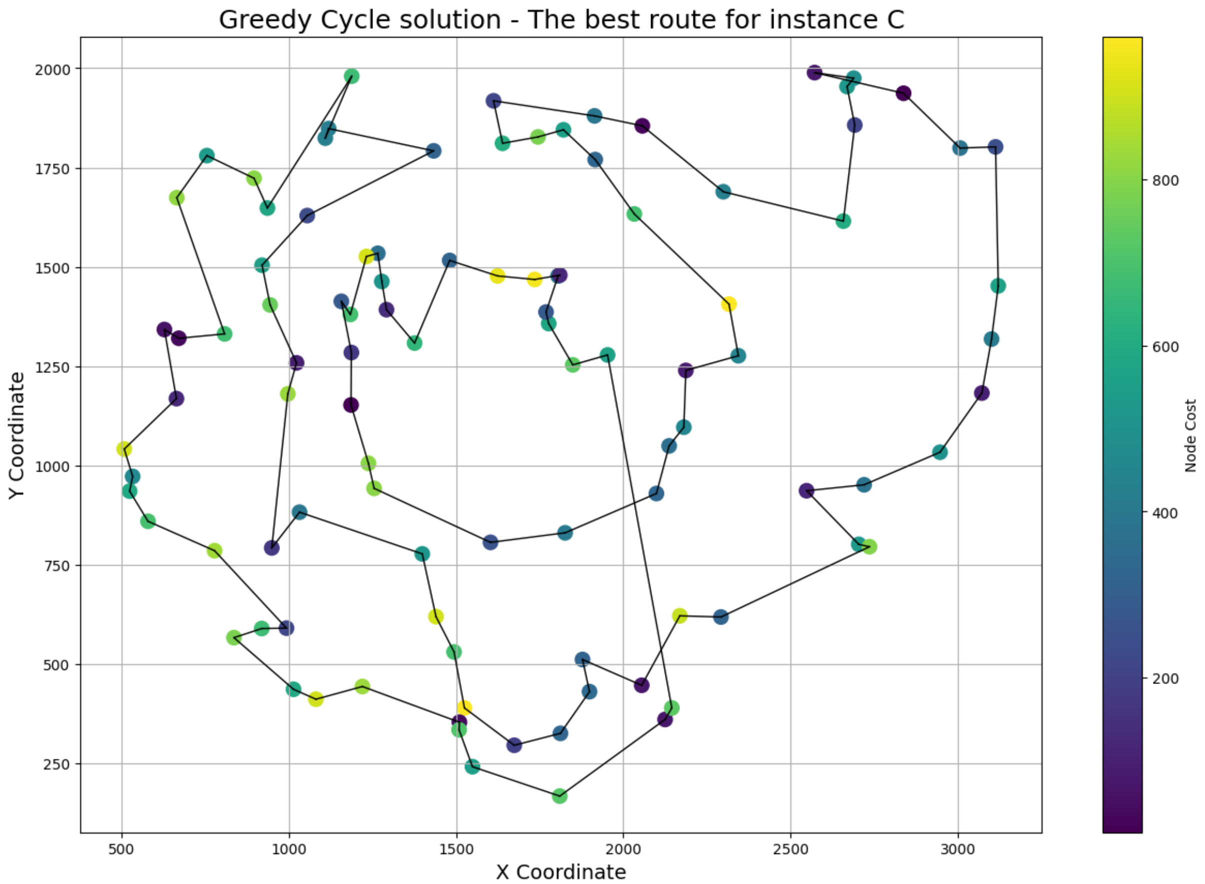
Nearest neighbor solution - The best route for instance D



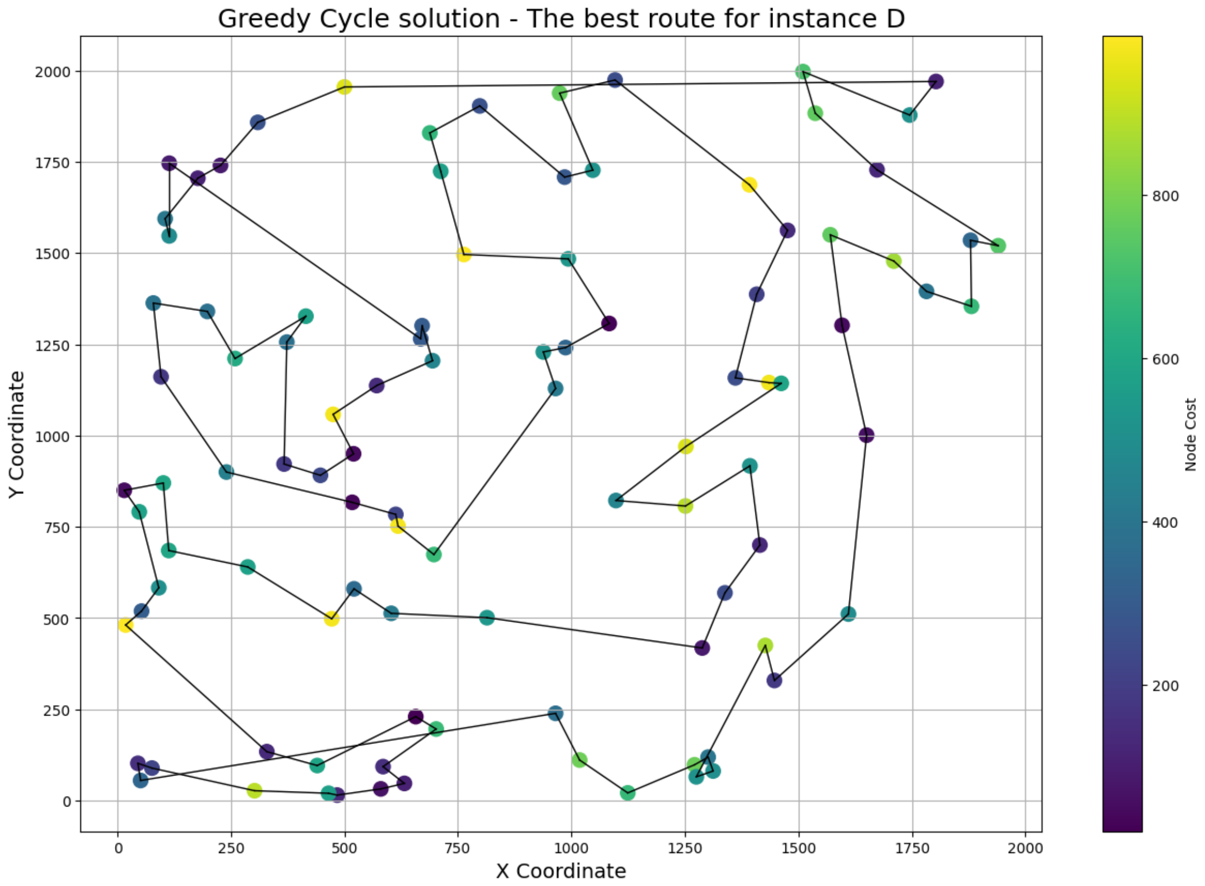
The greedy cycle algorithm







Greedy Cycle solution - The best route for instance C
Y Coordinate
X Coordinate
Node Cost



Greedy Cycle solution - The best route for instance D
Y Coordinate
X Coordinate
Node Cost