Evolutionary Computation

Assignment 1

Antoni Lasik

Daniel Jankowski

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 |

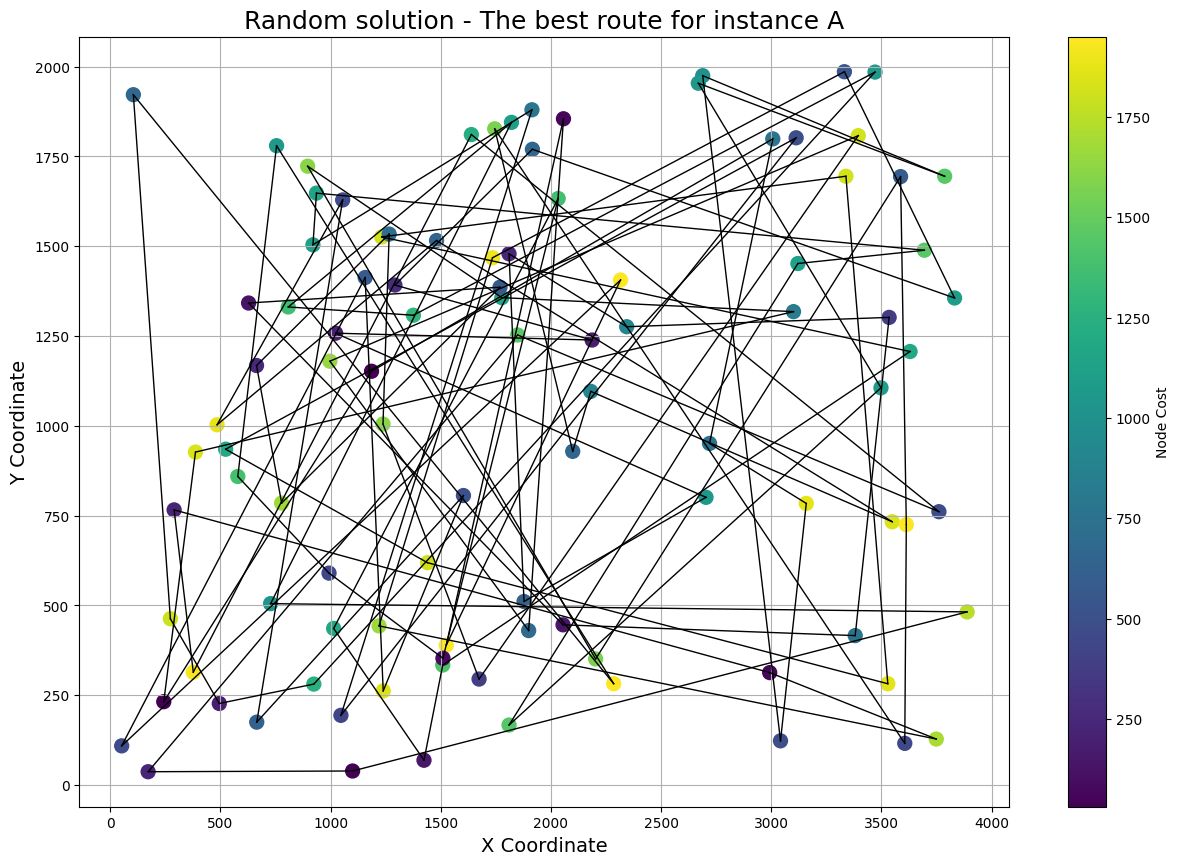
**The nearest neighbor algorithm**

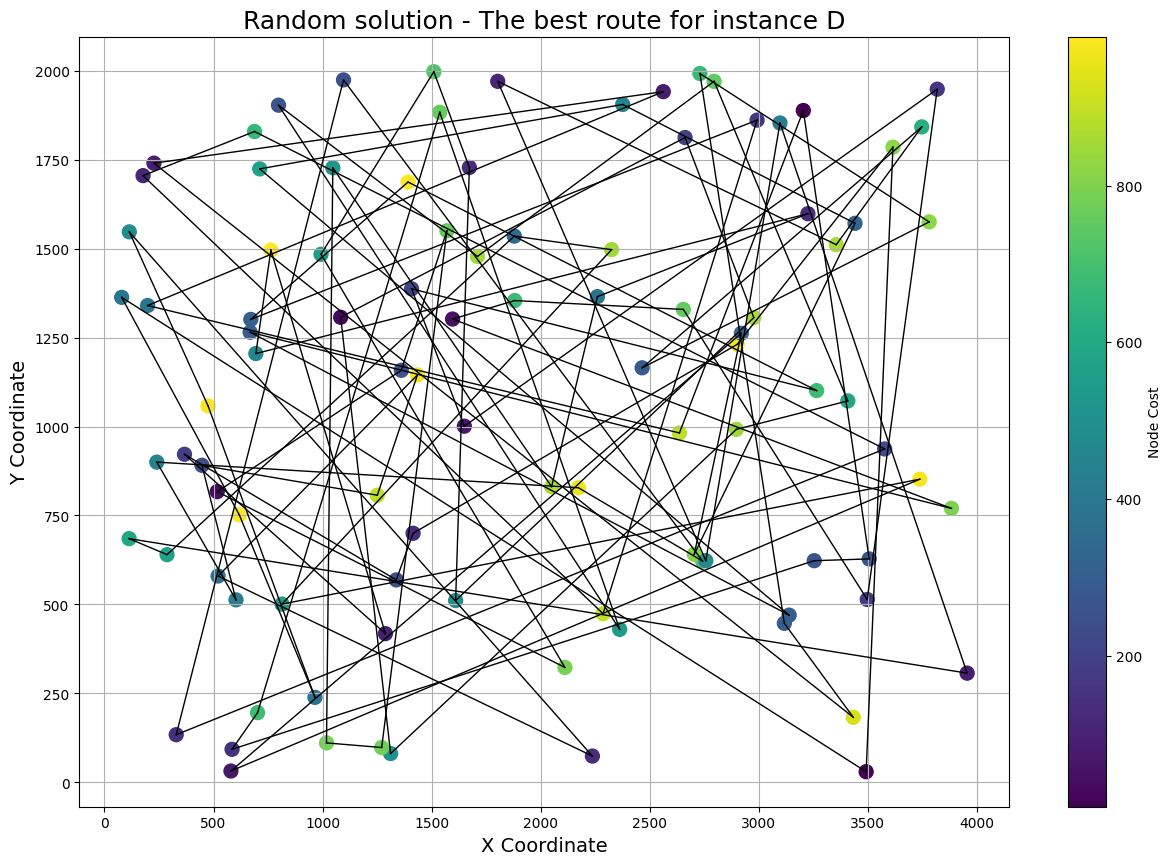
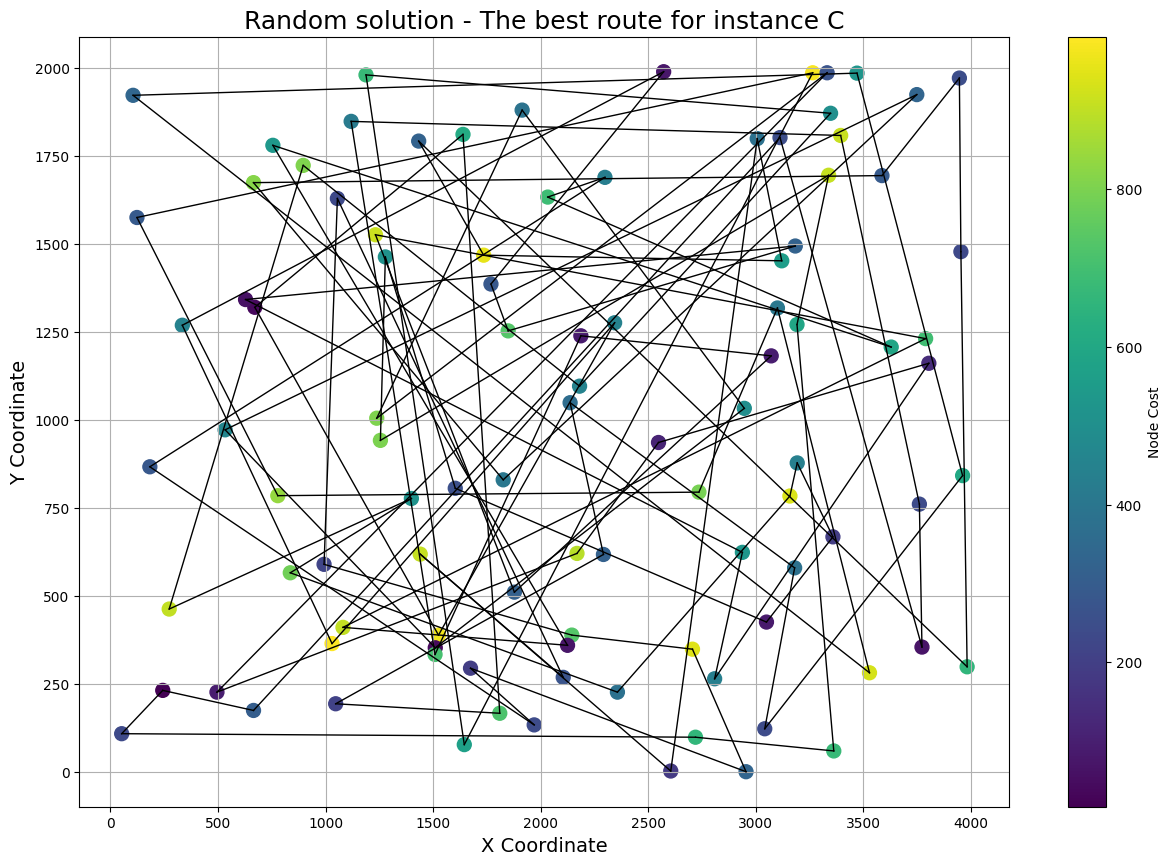
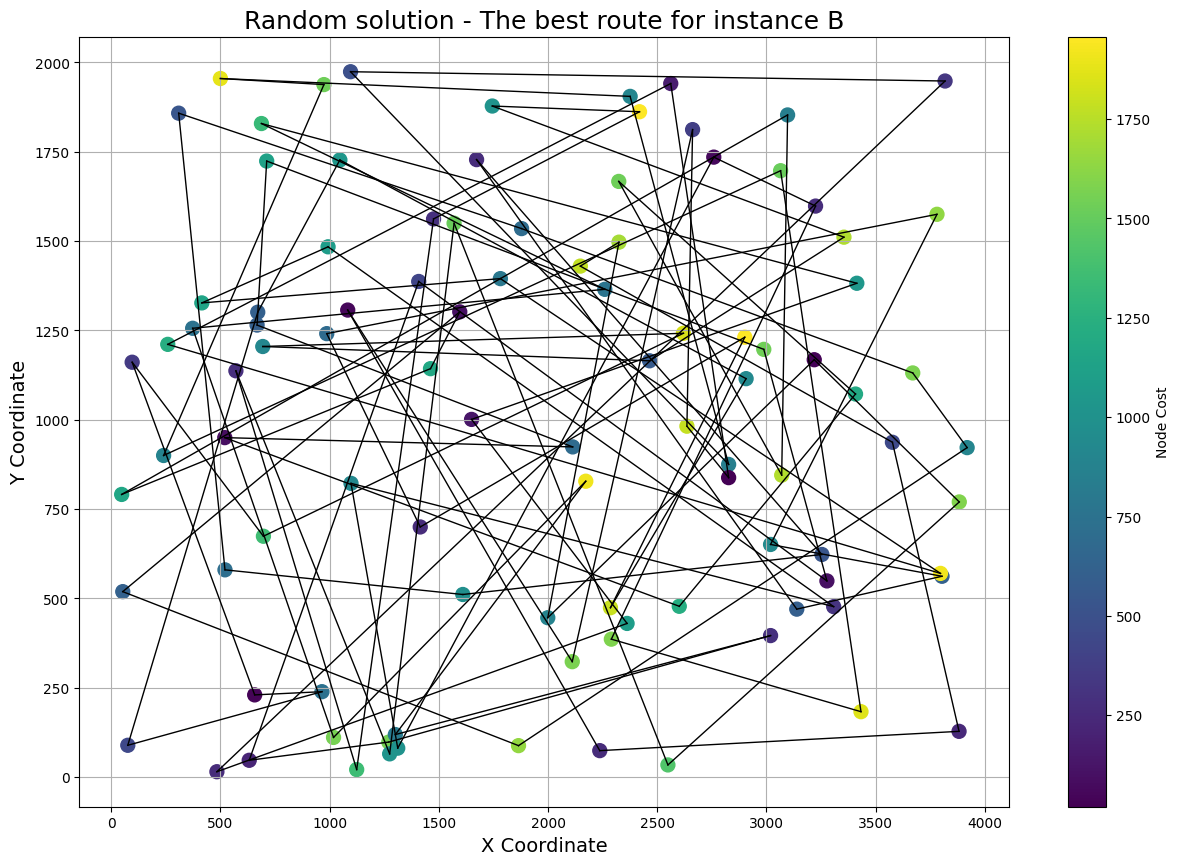
|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Max** | **Mean** |
| **Instance A** | 110,035 | 125,805 | 116,516.55 |
| **Instance B** | 109,047 | 124,759 | 116,413.93 |
| **Instance C** | 62,629 | 71,814 | 66,329.95 |
| **Instance D** | 62,967 | 71,396 | 67,119.2 |

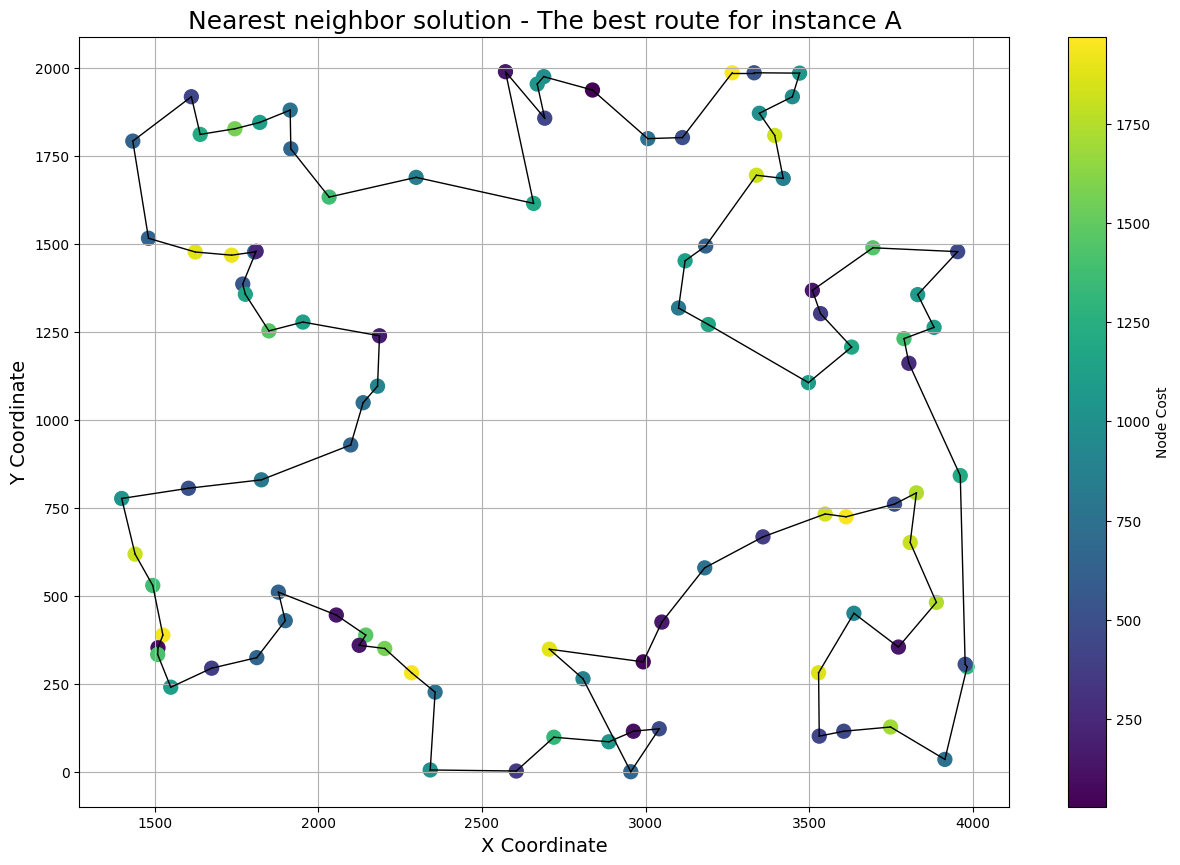
**The greedy cycle algorithm**

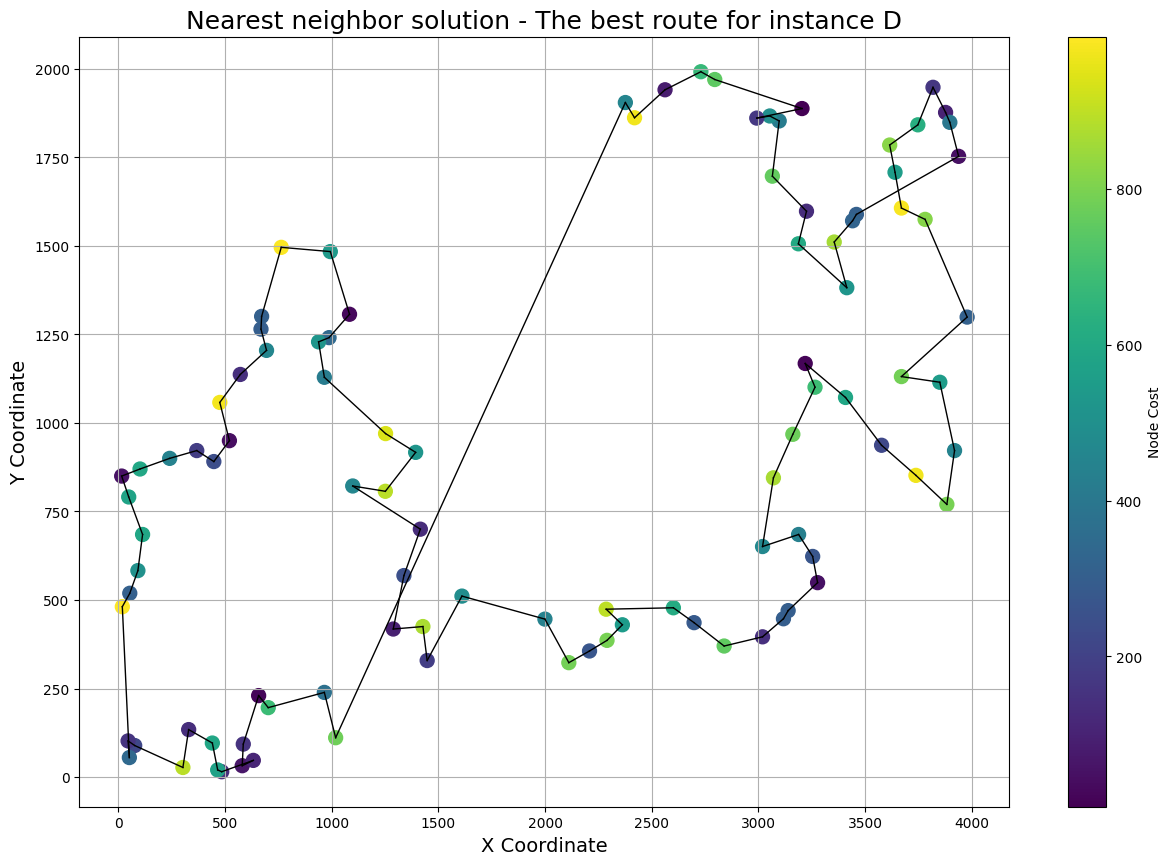
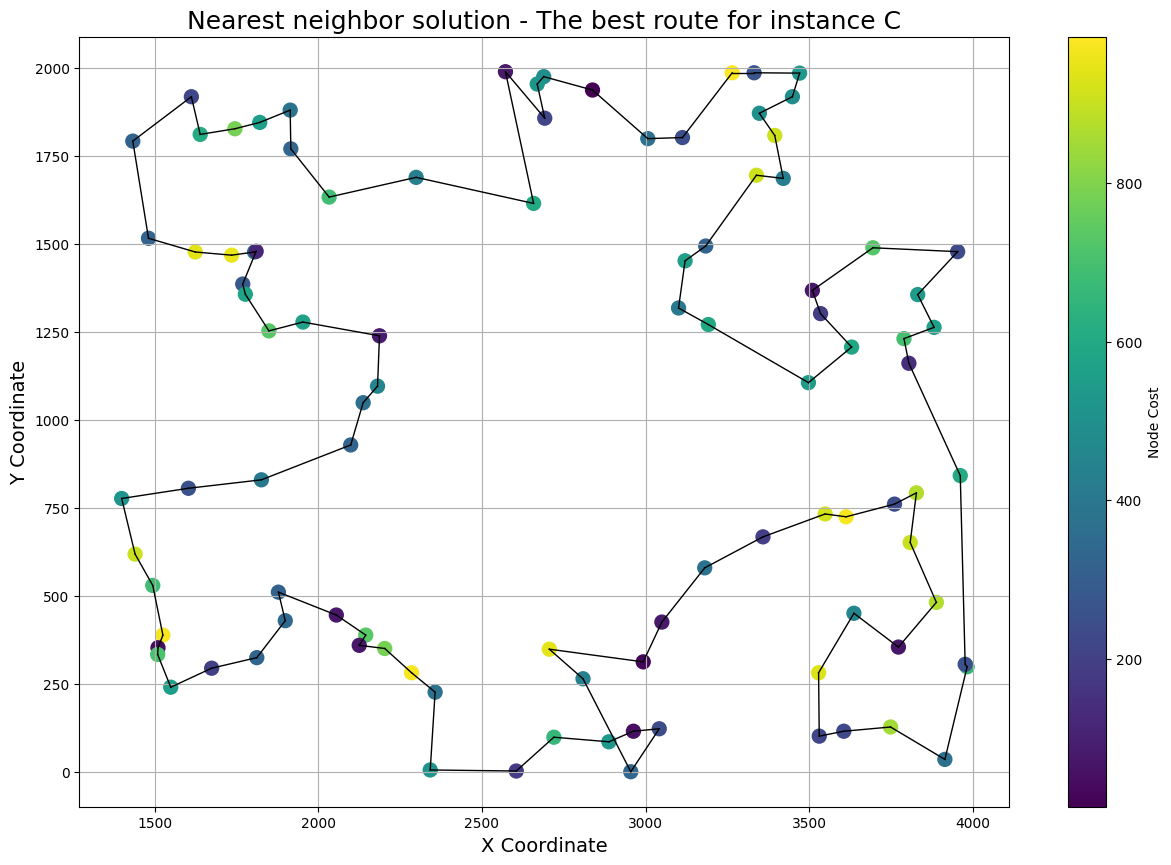
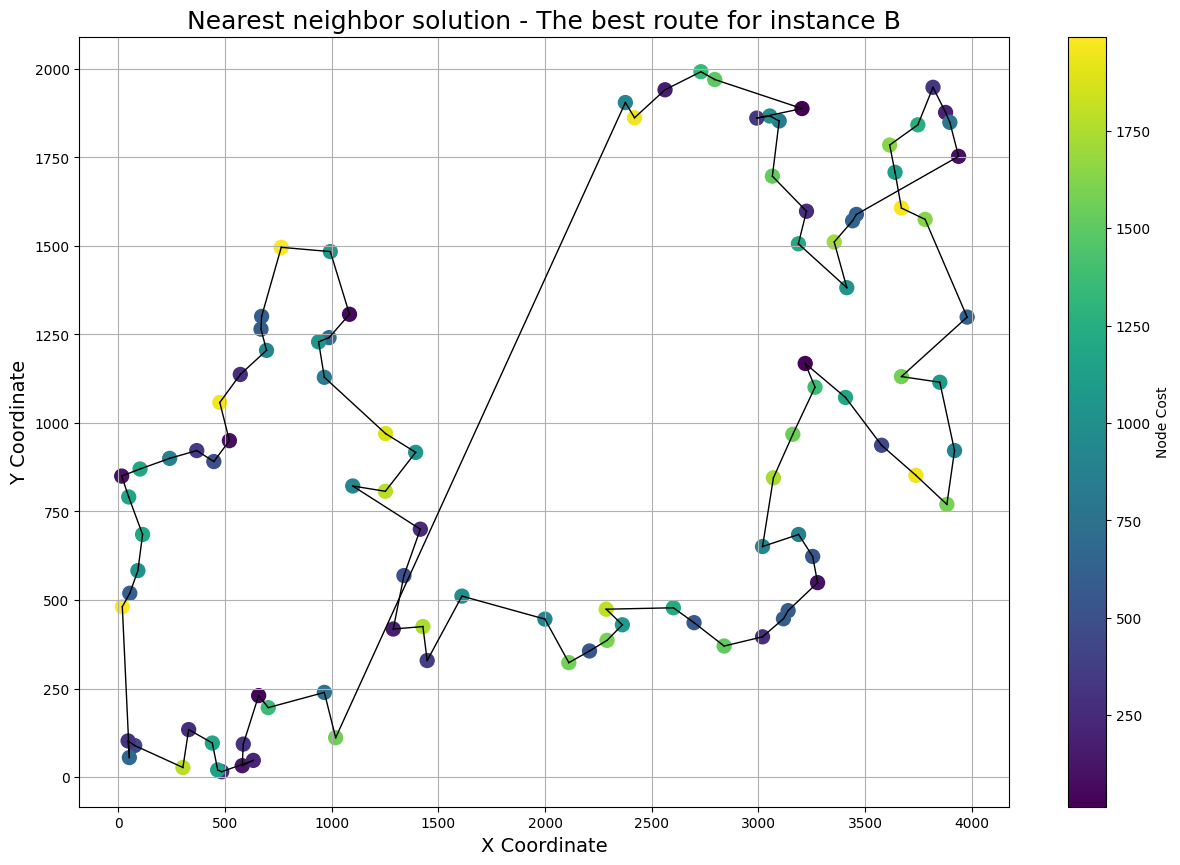
|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Max** | **Mean** |
| **Instance A** | 111,129 | 125,930 | 120,306.18 |
| **Instance B** | 108,501 | 129,321 | 119,611.18 |
| **Instance C** | 65,218 | 73,928 | 70,146.16 |
| **Instance D** | 63,694 | 74,892 | 69,988.40 |

**Visualizations**

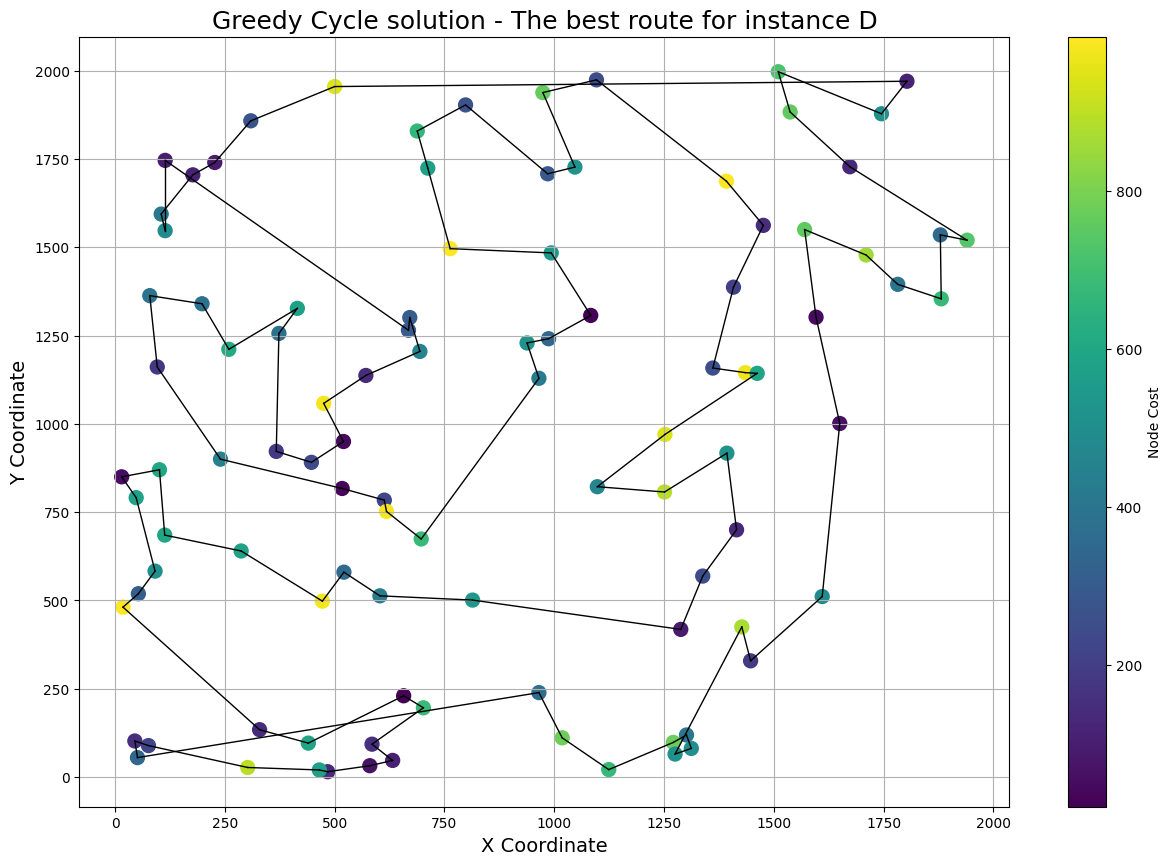
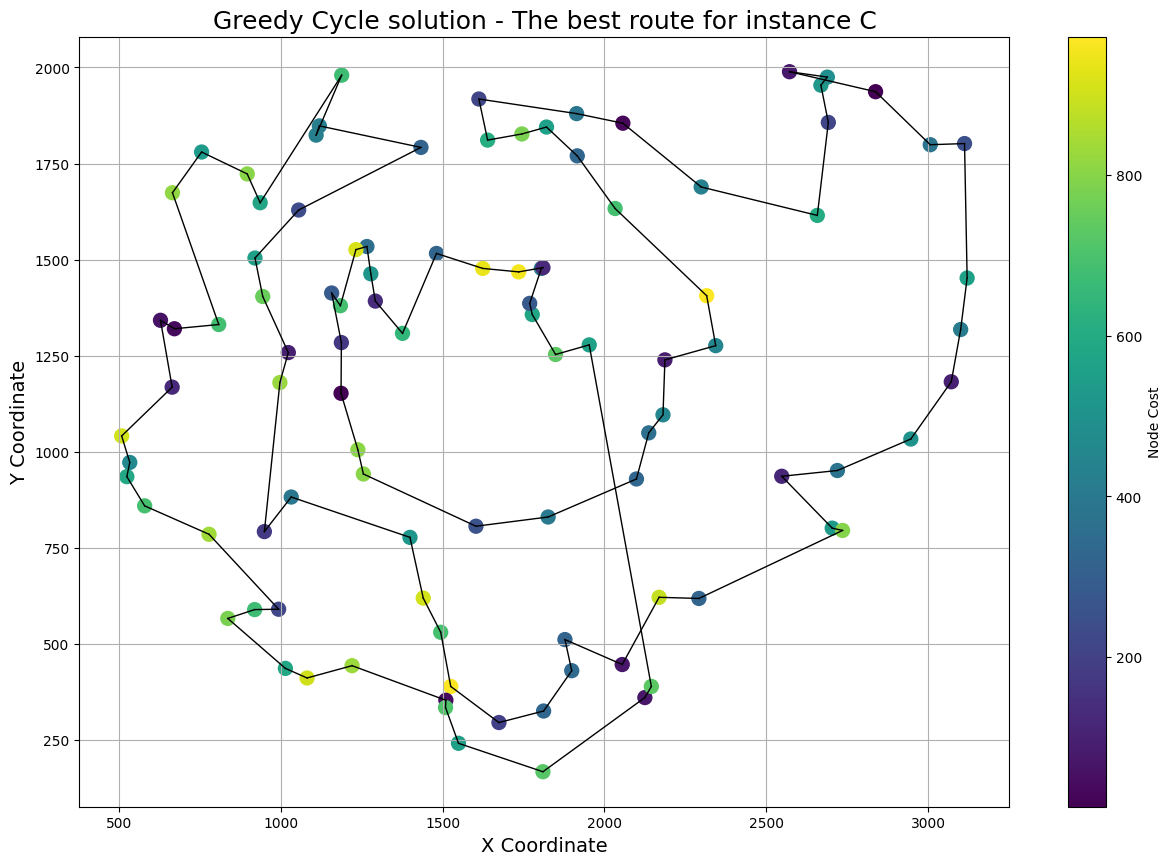
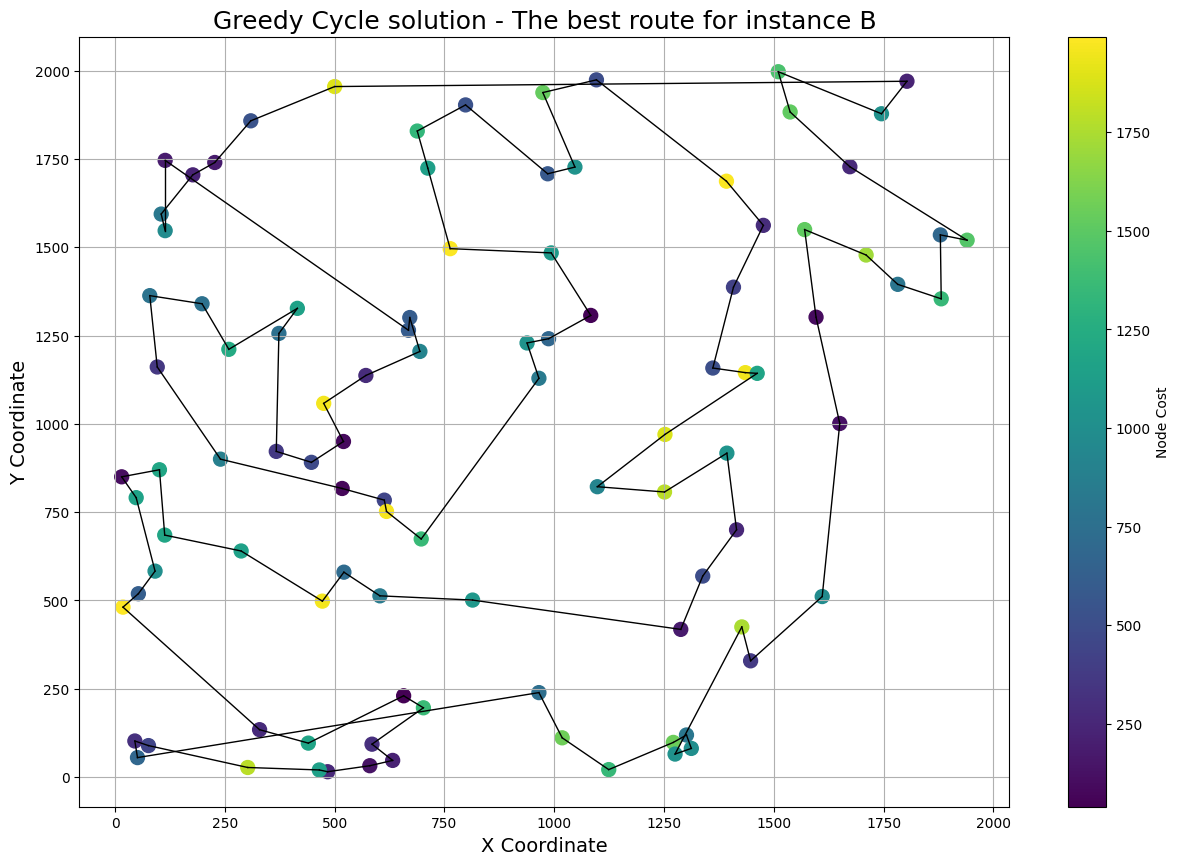
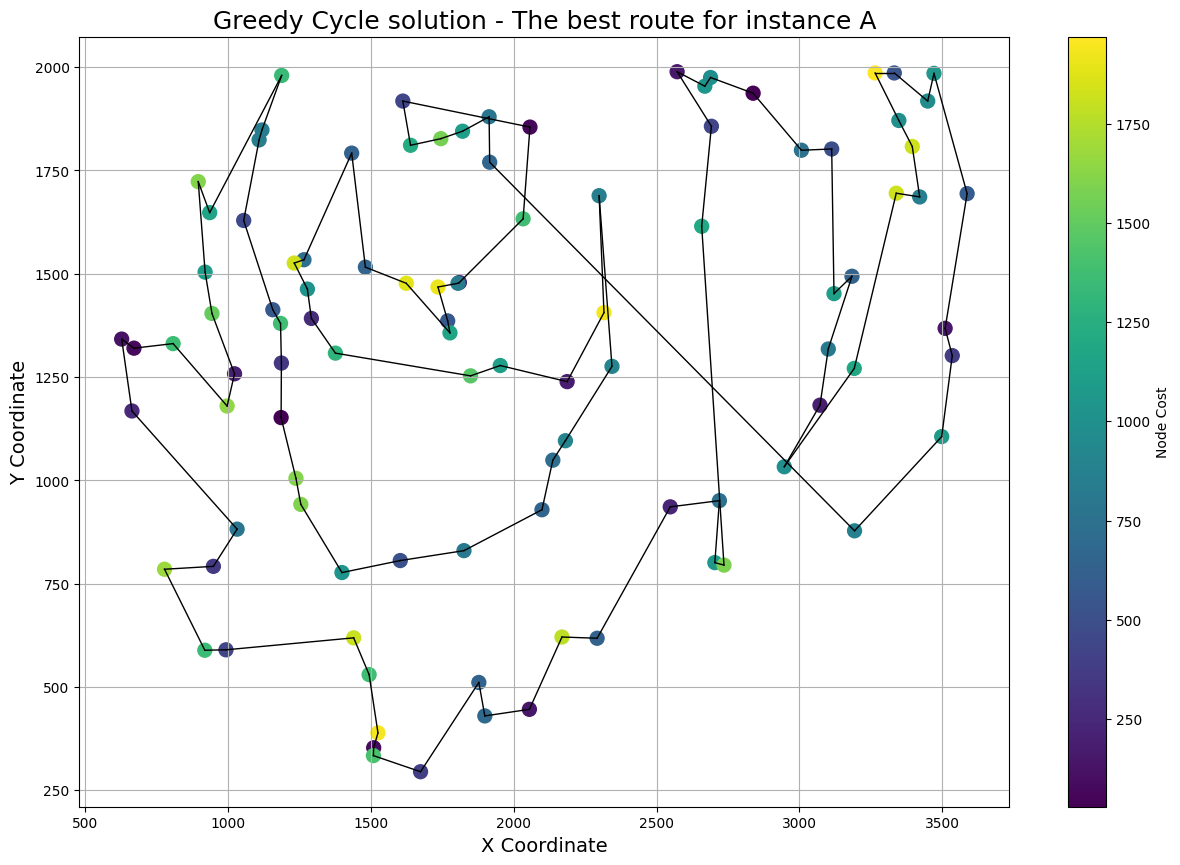
**The random solution**

**

**The nearest neighbor algorithm**

****

**The greedy cycle algorithm**

****

**Conclusions**