Evolutionary Computation

Assignment 2

Antoni Lasik 148287

Daniel Jankowski 148257

Source code: <https://github.com/JankowskiDaniel/evolutionary-computation/tree/main/assignment2>

**Problem description**

The task involves analyzing three columns of integers, each row corresponding to a single node. The initial two columns designate the x and y coordinates, pinpointing the nodes' locations on a plane, while the third column specifies the cost associated with each node. The objective is to meticulously choose an exact half of the total nodes (in cases where the total node count is an odd number, the count of nodes to be selected is adjusted upward to the nearest whole number) to construct a Hamiltonian cycle, which is essentially a continuous loop that passes through each member of the selected set of nodes. The criterion for this selection is that the aggregate of the complete path's length and the cumulative cost of the chosen nodes should be as low as possible.

To quantify the distances between nodes, we employ the Euclidean distance formula, and the resulting figures are rounded off to the nearest integer in a standard mathematical fashion. Moreover, as part of the distance between nodes, we take into account the cost of the destination node. This ensures that cost has a significant impact on the final results.

We implement two heuristic methods based on the greedy cycle algorithm: greedy 2-regret heuristics and greedy heuristics with a weighted sum criterion.

**Pseudocode of implemented algorithms**

**An additional method for calculating the distance matrix**

***Calculate\_distance\_matrix(coords, costs):***

*dist\_matrix = [][]*

***FOR*** *i in range(len(coords)):*

***FOR*** *j in range(len(coords)):*

*dist\_matrix[i][j] = round(sqrt((coords[i].x – coords[j].x)\*\*2 + (coords[i].y – coords[j].y)\*\*2) + costs[j]*

***return*** *dist\_matrix*

**Greedy 2-regret heuristic**

**Generate\_greedy\_2regret\_solution(dist\_matrix, 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:*

*regret\_node = None*

*regret\_position = None*

*best\_regret= float('-inf')*

*regret\_best\_increase = float('inf')*

***FOR*** *node in unselected\_nodes:*

*best\_node = None*

*best\_position = None*

*best\_min\_increase =infinity*

*second\_best\_min\_increase = infinitiy*

***FOR*** *i in range(len(selected\_nodes):*

*next\_index = (i +1)% len(selected\_nodes)*

*increase = (dist\_matrix[selected\_nodes[i], node] +*

*dist\_matrix[node, selected\_nodes[next\_i]] -*

*dist\_matrix[selected\_nodes[i], selected\_nodes[next\_i]])*

*// calculate the increase of distance if the node is inserted after the node of index i*

***IF*** *increase < second\_best\_min\_increase:*

***IF*** *increase < best\_min\_increase:*

*best\_min\_increase = increase*

*best\_node = node*

*best\_position = next\_i*

***ELSE****:*

*second\_best\_min\_increase = increase*

*regret= second\_best\_min\_increase - best\_min\_increase*

***IF*** *regret > best\_regret:*

*best\_regret = regret*

*regret\_node = best\_node*

*regret\_position = best\_position*

*regret\_best\_increase = best\_min\_increase*

***ADD*** *regret\_node at the regret\_position to selected nodes*

***REMOVE*** *regret\_node from unselected\_nodes total\_distance += regret\_best\_increase*

*total\_distance += dist\_matrix[selected\_nodes[-1], selected\_nodes[0]]*

***return*** *selected\_nodes, total\_distance*

**Greedy 2-regret with a weighted sum**

**Generate\_greedy\_2regret\_weights\_solution(dist\_matrix, start\_node, weight):**

*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:*

*regret\_node = None*

*regret\_position = None*

*best\_regret= float('-inf')*

*regret\_best\_increase = float('inf')*

***FOR*** *node in unselected\_nodes:*

*best\_node = None*

*best\_position = None*

*best\_min\_increase =infinity*

*second\_best\_min\_increase = infinitiy*

***FOR*** *i in range(len(selected\_nodes):*

*next\_index = (i +1)% len(selected\_nodes)*

*increase = (dist\_matrix[selected\_nodes[i], node] +*

*dist\_matrix[node, selected\_nodes[next\_i]] -*

*dist\_matrix[selected\_nodes[i], selected\_nodes[next\_i]])*

*// calculate the increase of distance if the node is inserted after the node of index i*

***IF*** *increase < second\_best\_min\_increase:*

***IF*** *increase < best\_min\_increase:*

*best\_min\_increase = increase*

*best\_node = node*

*best\_position = next\_i*

***ELSE****:*

*second\_best\_min\_increase = increase*

*regret= second\_best\_min\_increase - best\_min\_increase*

*score = weight \* regret - (1-weight)\*best\_min\_increase*

***IF*** *score > best\_score:*

*best\_score = score*

*score\_node = best\_node*

*score\_position = best\_position*

*score\_best\_increase = best\_min\_increase*

***ADD*** *score\_node at the score\_position to selected nodes*

***REMOVE*** *score\_node from unselected\_nodes total\_distance += score\_best\_increase*

*total\_distance += dist\_matrix[selected\_nodes[-1], selected\_nodes[0]]*

***return*** *selected\_nodes, total\_distance*

**Results**

**Results from the previous report**

**The random solution**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Max** | **Mean** |
| **Instance A** | 237,941 | 288,302 | 264,028.49 |
| **Instance B** | 243,288 | 295,269 | 266,655.16 |
| **Instance C** | 191,705 | 241,451 | 214,929.07 |
| **Instance D** | 191,218 | 242,515 | 219,678.85 |

**The nearest neighbor algorithm**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Max** | **Mean** |
| **Instance A** | 84,471 | 95,013 | 87,679.14 |
| **Instance B** | 77,448 | 82,631 | 79,282.58 |
| **Instance C** | 56,304 | 63,697 | 58,872.68 |
| **Instance D** | 50,335 | 59,846 | 54,290.68 |

**The greedy cycle algorithm**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Max** | **Mean** |
| **Instance A** | 75,136 | 80,025 | 76,711.19 |
| **Instance B** | 67,896 | 76,096 | 70,464.27 |
| **Instance C** | 53,020 | 58,499 | 55,859.31 |
| **Instance D** | 50,288 | 60,208 | 54,931.05 |

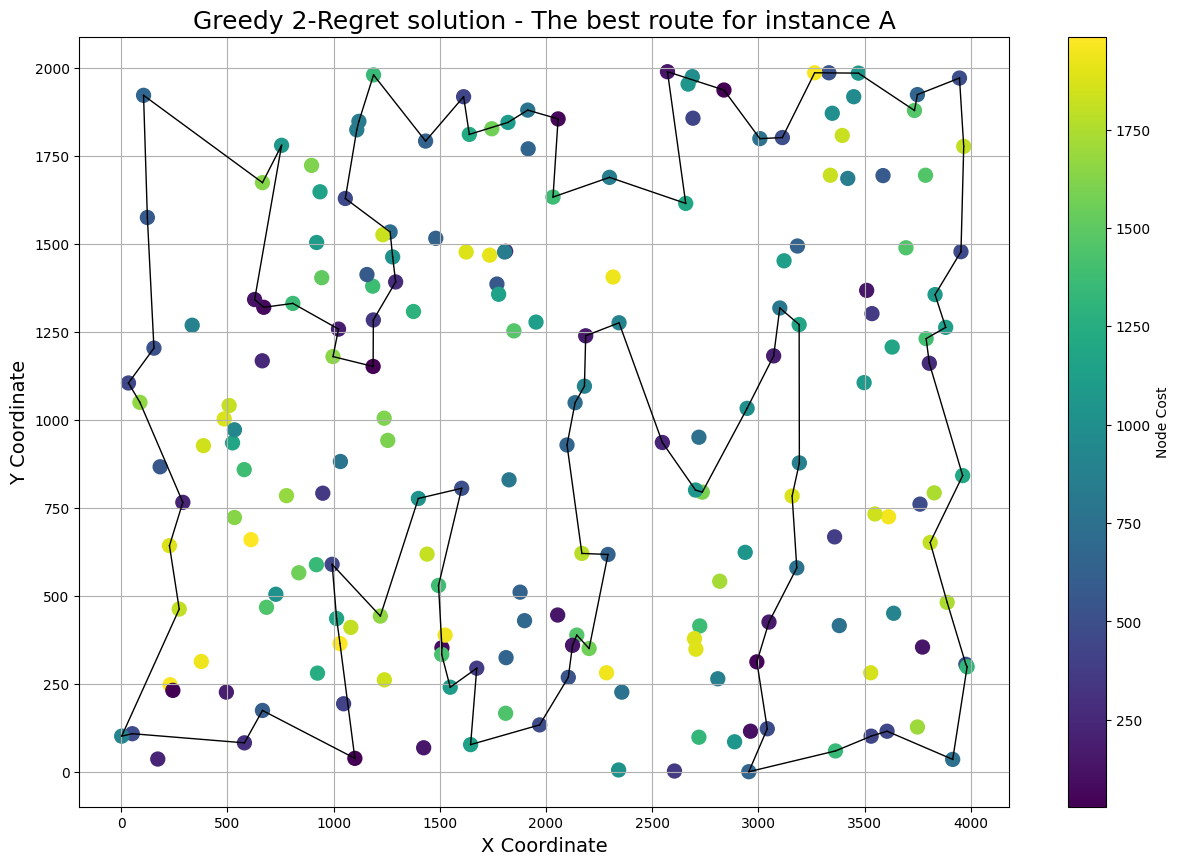
**The greedy 2-regret algorithm**

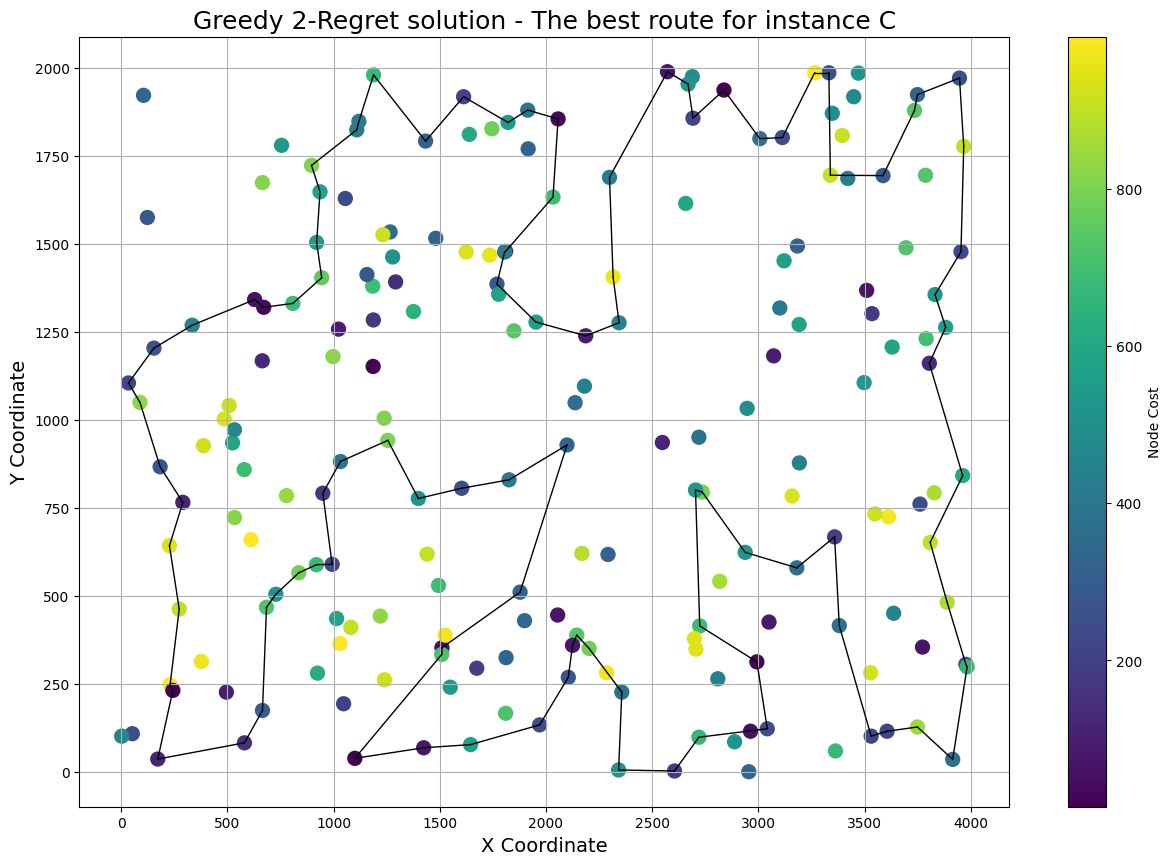
|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Max** | **Mean** |
| **Instance A** | 106,734 | 124,404 | 116,772.94 |
| **Instance B** | 104,997 | 125,925 | 116,871.66 |
| **Instance C** | 63,247 | 72,558 | 68,444.90 |
| **Instance D** | 62,852 | 74,184 | 68,585.68 |

**The greedy 2-regret with a weighted sum**

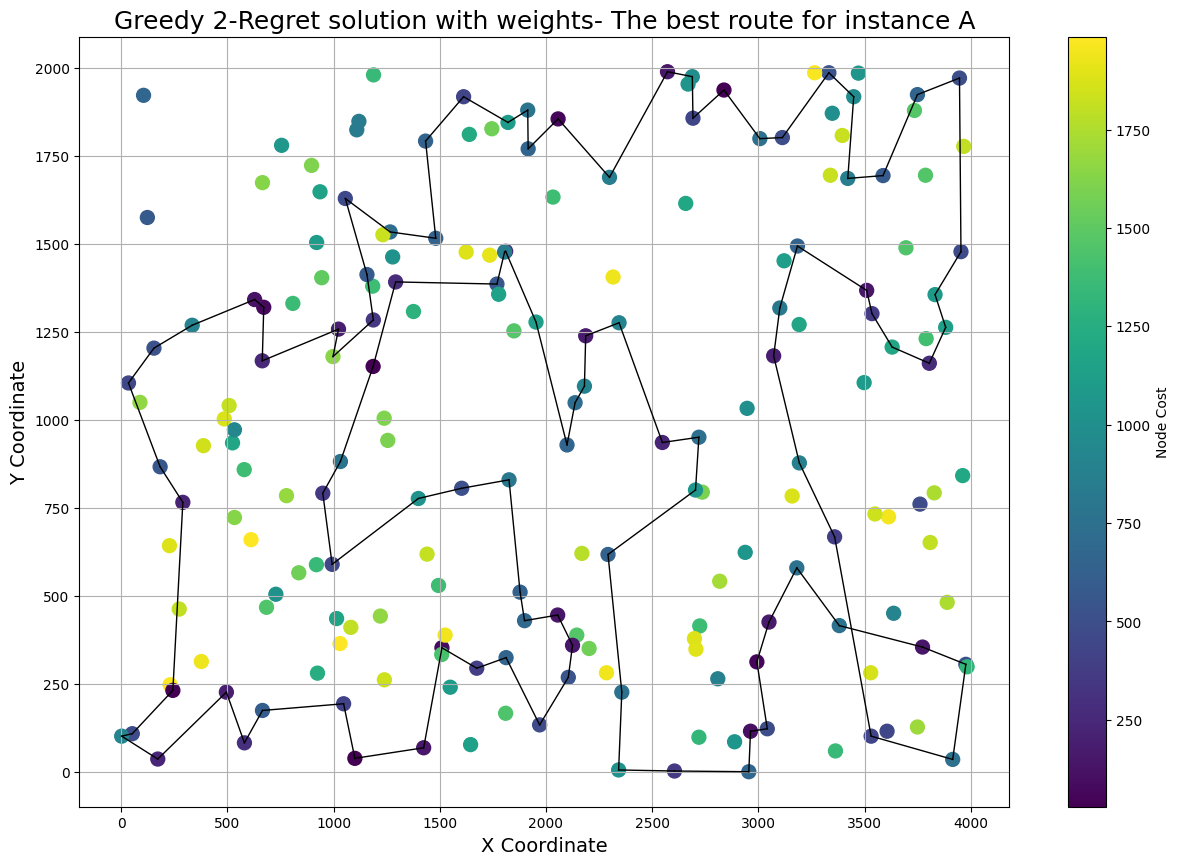
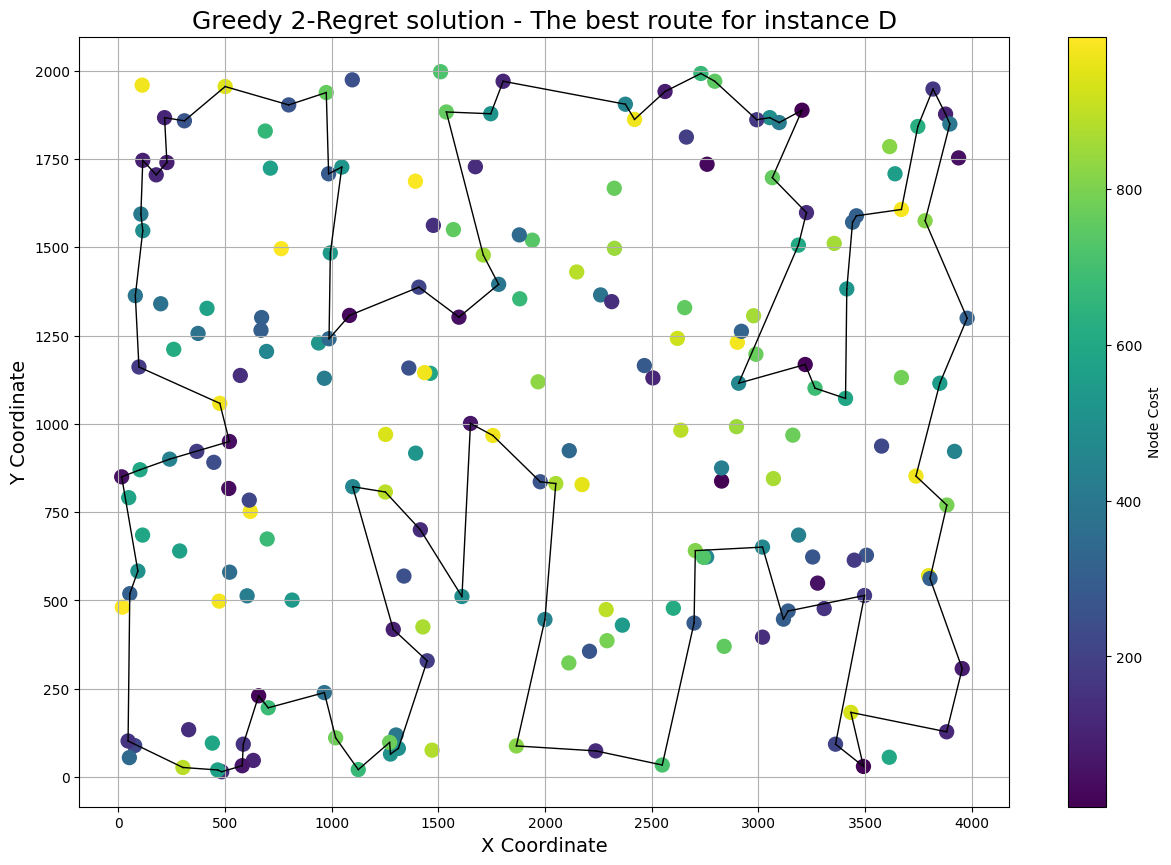
|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Max** | **Mean** |
| **Instance A** | 74,708 | 82,990 | 76,980.75 |
| **Instance B** | 67,490 | 80,001 | 73,067.76 |
| **Instance C** | 50,158 | 58,173 | 53,795.98 |
| **Instance D** | 46,549 | 62,321 | 52,930.70 |

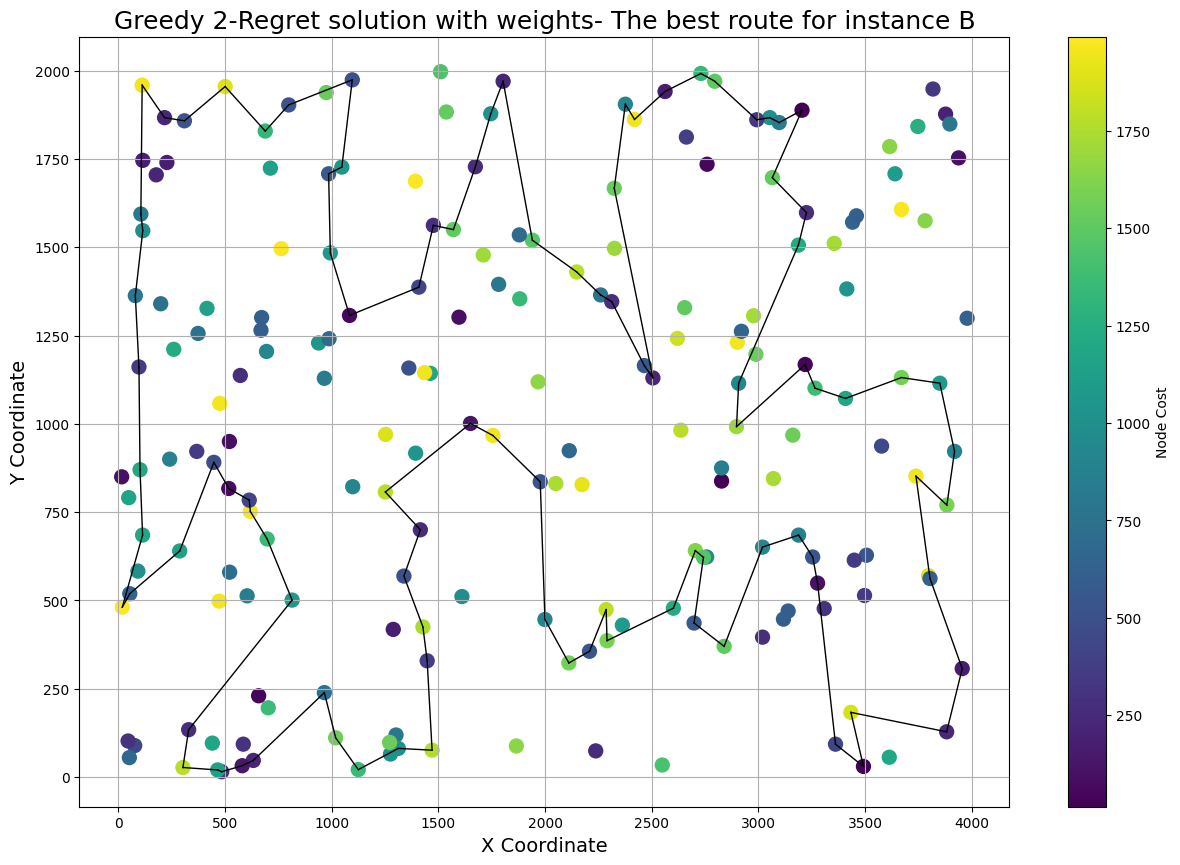
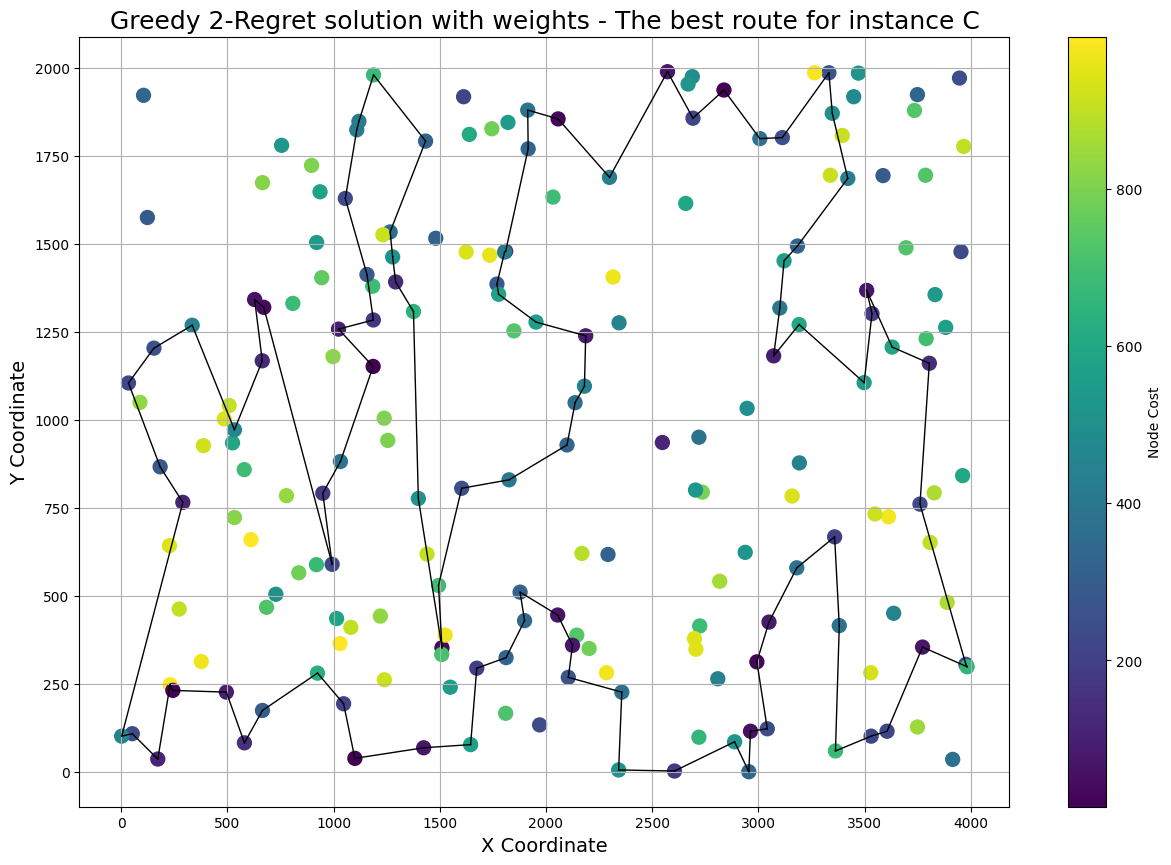
**Visualizations**

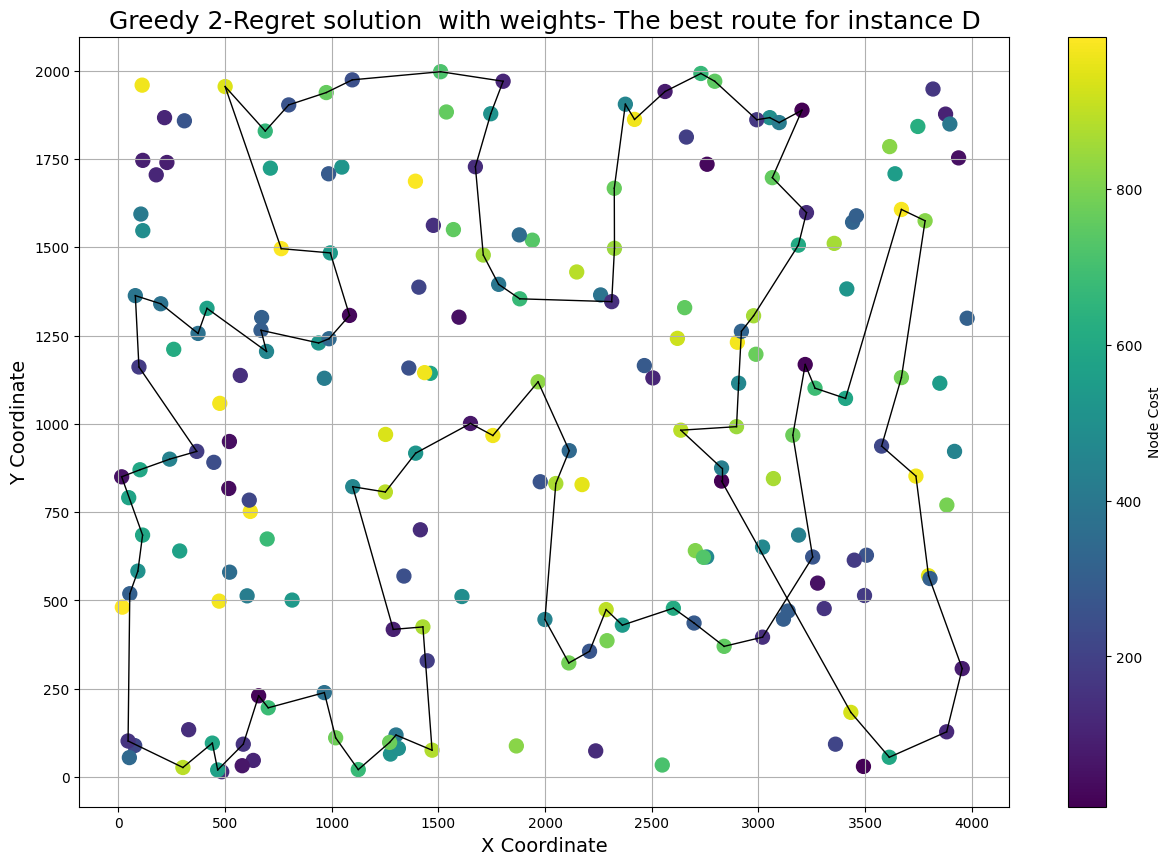
**The greedy 2-regret algorithm**

**A graph of a graph

Description automatically generated with medium confidence**

******The greedy 2-regret with a weighted sum**



**Conclusions**

The analysis of the heuristic algorithms applied to the Traveling Salesman Problem (TSP) reveals significant differences in performance, underscoring the importance of algorithm selection in obtaining efficient solutions. The Random Solution, devoid of any strategic path selection, resulted in the least efficient routes, emphasizing the necessity for heuristic logic in tackling such optimization problems. The Nearest Neighbor and Greedy Cycle algorithms demonstrated considerable improvements in comparison to the random algorithm. Especially the greedy cycle where paths started to cross each other much less than for NN and random algorithm.

Interestingly, the Greedy 2-Regret algorithm, despite its strategy to minimize future regret, didn't always yield shorter paths. This can be attributed to its inherent mechanism of considering not just the immediate distance but also potential future path restrictions, reflecting the trade-offs involved in decision-making during optimization problems. However, the most efficient paths were often produced by the Greedy 2-Regret algorithm with a weighted sum, indicating that a more holistic approach—incorporating additional factors like node costs into the decision metric—can lead to superior solutions. This algorithm's ability to balance immediate distance with overall route cost and regret resulted in paths that, while not always the shortest, were often the most cost-effective.

The visualizations presented above show more direct and less convoluted paths for the more advanced heuristics, especially the Greedy 2-Regret with a weighted sum. These findings highlight the critical role of sophisticated heuristics in solving complex optimization problems like the TSP, where multiple factors must be considered beyond mere distance. They also underscore the potential for even more advanced algorithms or hybrid approaches that could further enhance solution efficiency, especially in real-world scenarios where variables such as time, cost, and resource constraints are of paramount importance.