**Algorithms and Data Structures**

Coursework 3 Report

**TSP Algorithms**

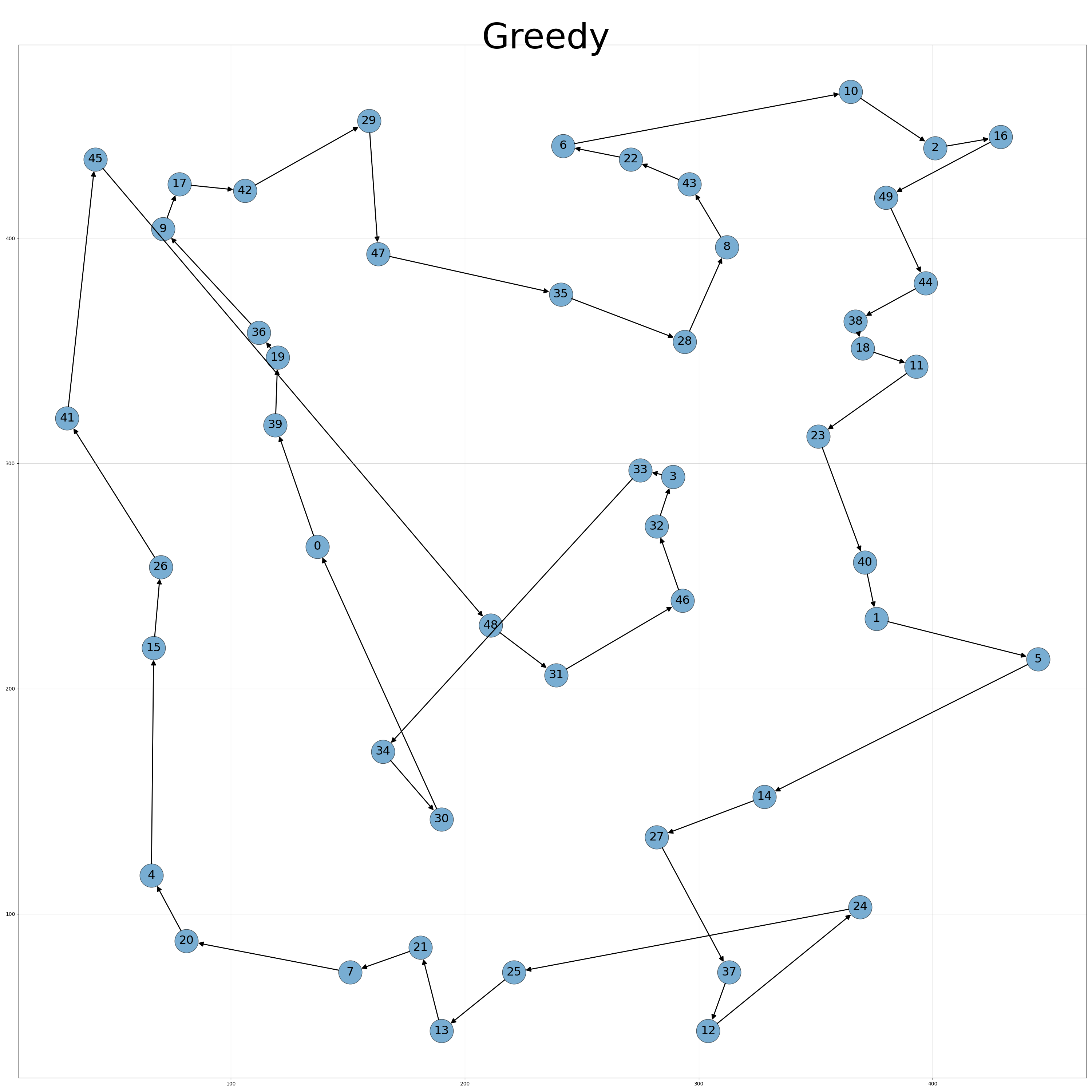
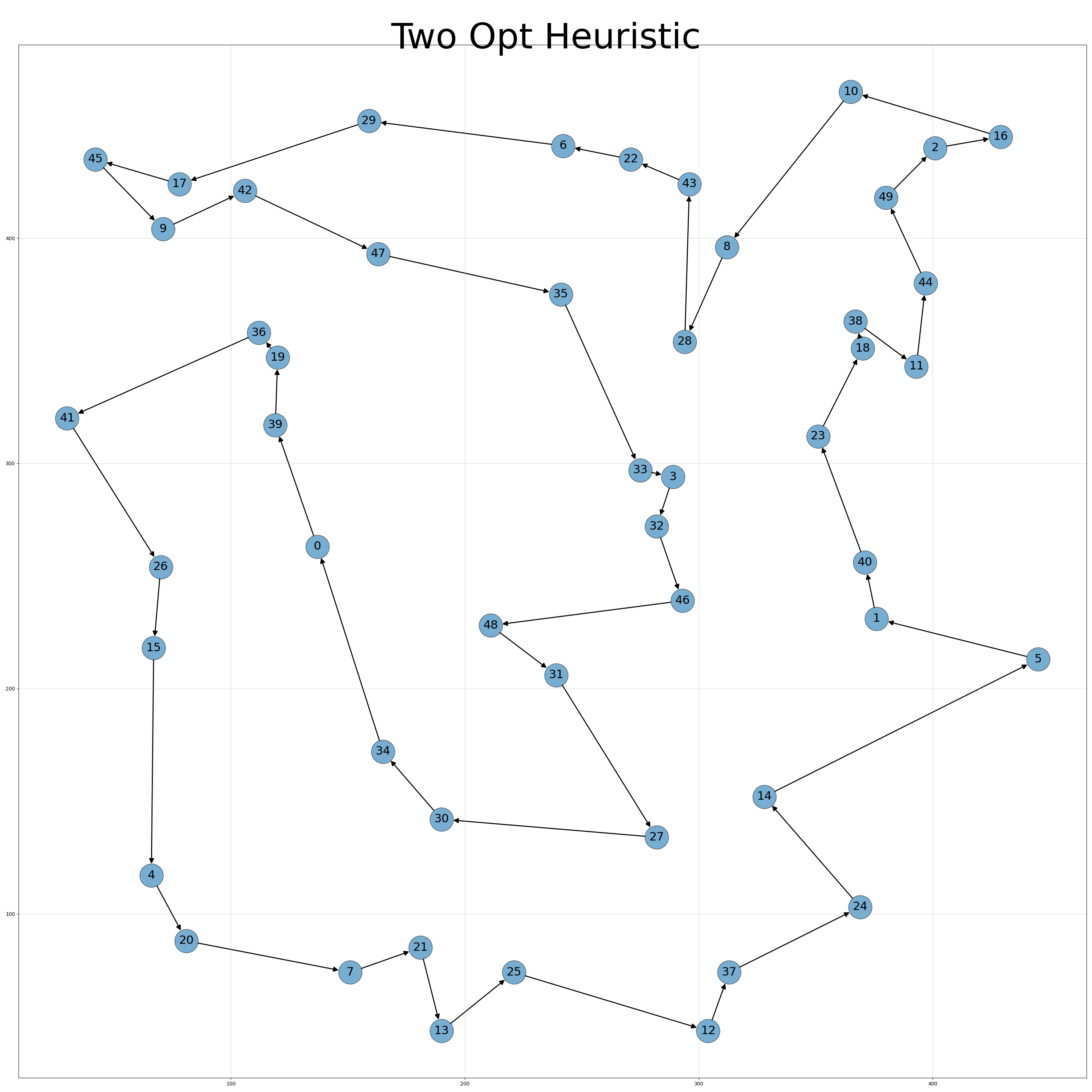
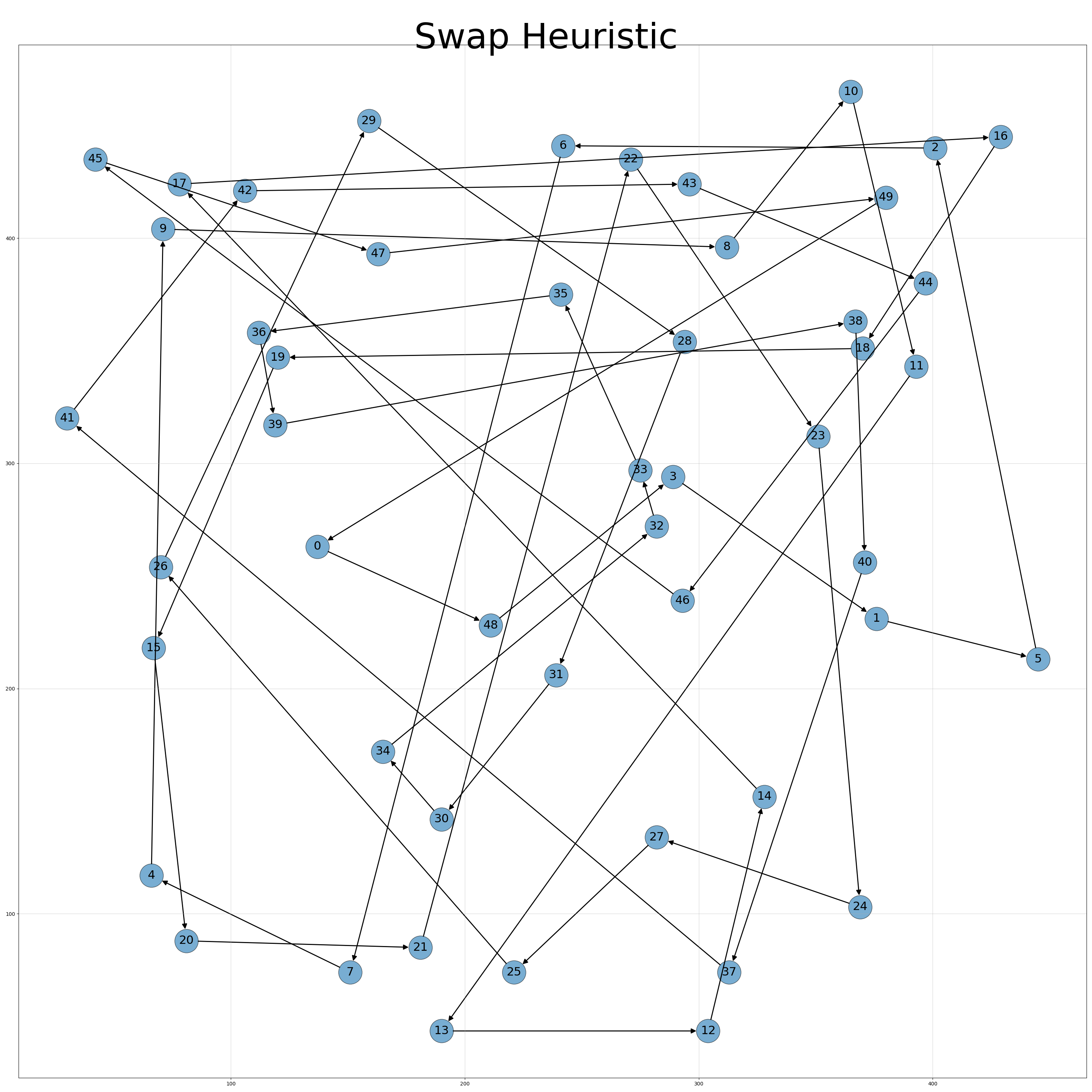
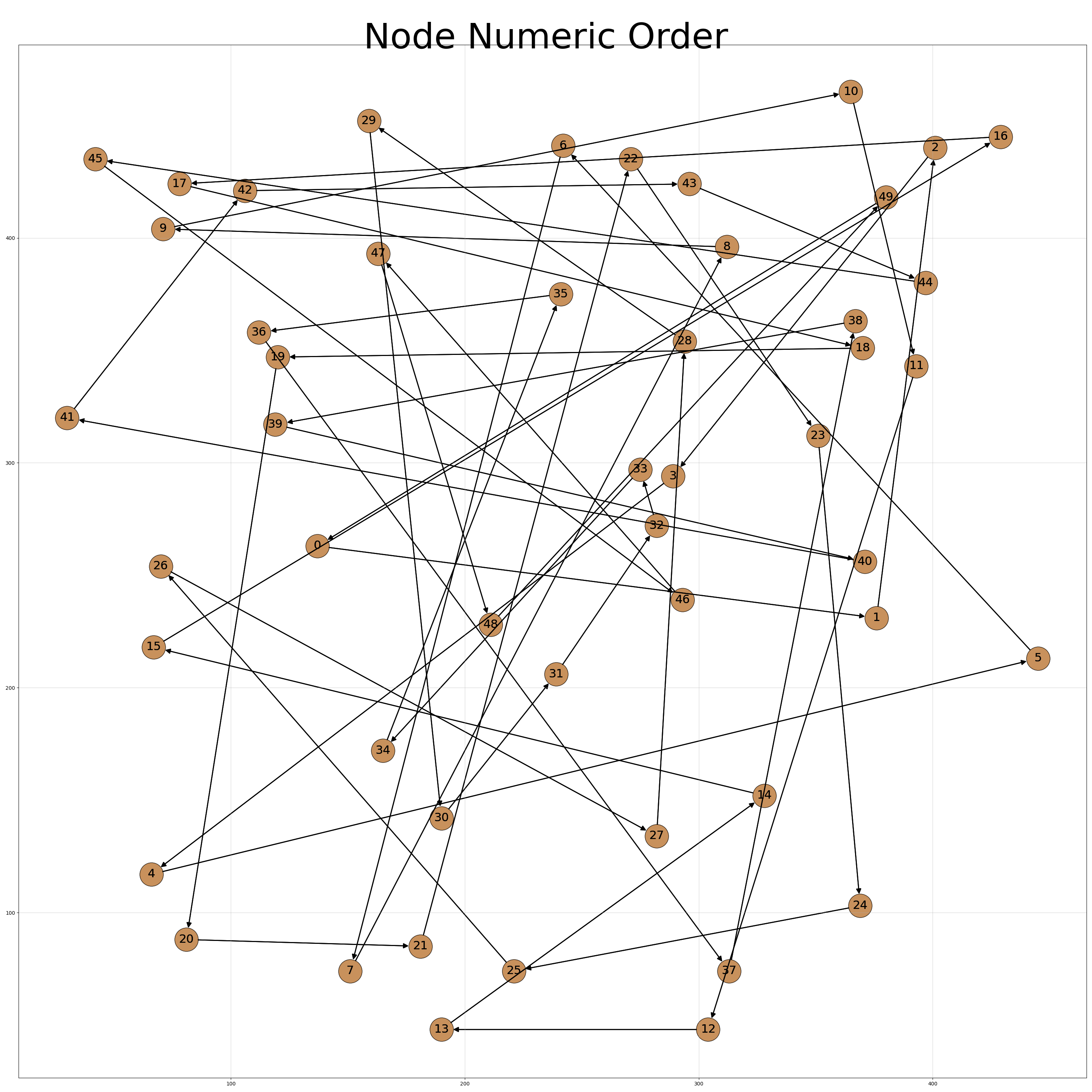
The travelling salesman problem (TSP) is a well-known problem, one which is NP-complete in its decision form. Therefore, an algorithm to reliably find the perfect solution would have to run in exponential time at best.

As such to find an answer to the solution in reasonable time, approximation algorithms must be used instead. 3 such methods of producing approximate solutions using heuristics have already been explored in the 2-opt, swap and greedy algorithms.

The swap and greedy algorithms are self-explanatory in their workings, using comparisons to adjacent vertices to shorten the length of the tour. The swap attempting to shorten the path by swapping adjacent vertices and the greedy selecting the next closest adjacent vertex

The 2-opt heuristic takes advantage of the Euclidean quality ensuring that a tour with paths crossing is invariably at least as long as the same tour with the paths uncrossed. By locally swapping the order of visiting any combination of two vertices in a tour, graphically, a cross in the paths will be caused or reversed at each attempted swap. Due to this, 2-opt performs very well in “real world” examples of the TSP.

The impacts of the algorithm on the tour are visualized below for the cities50 dataset.



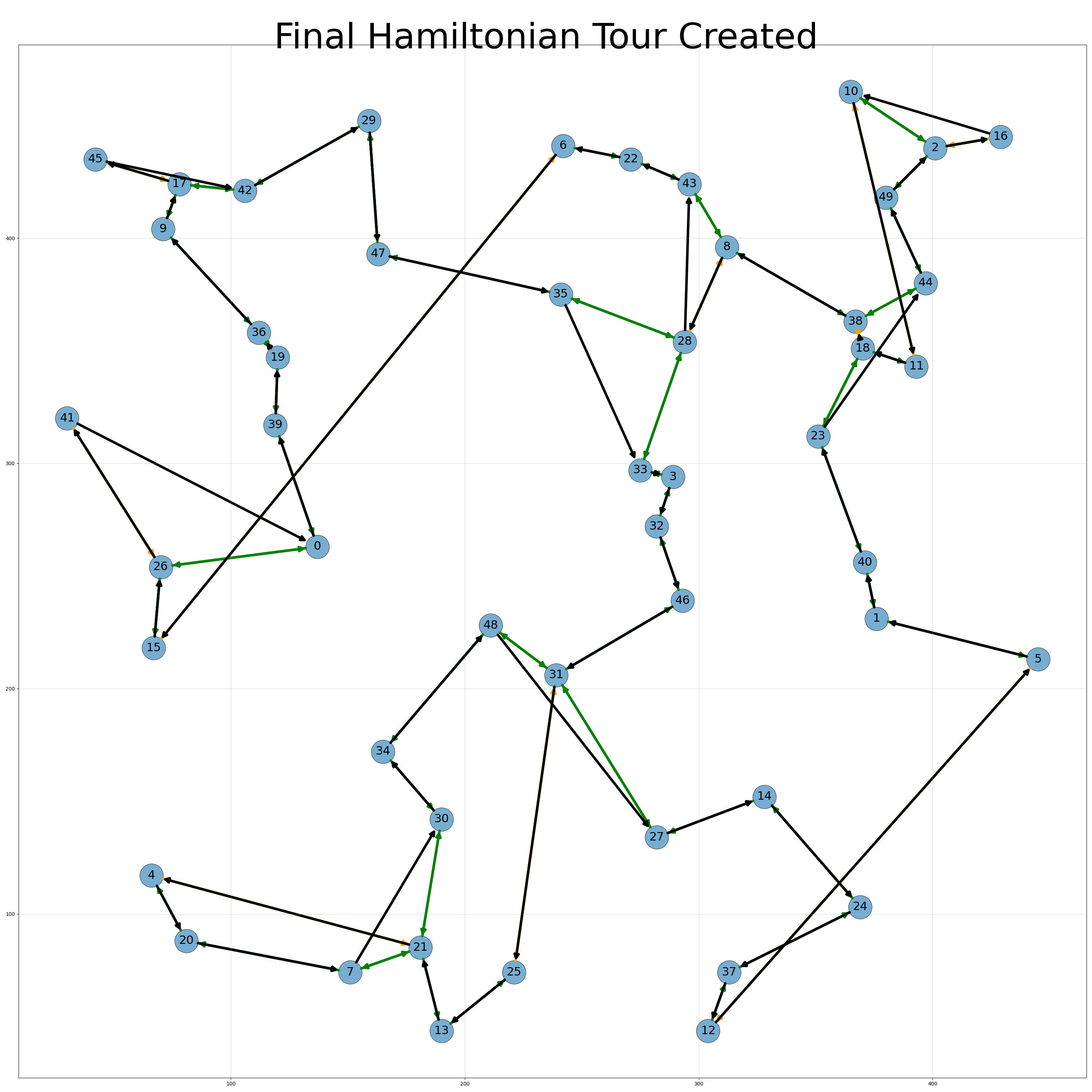
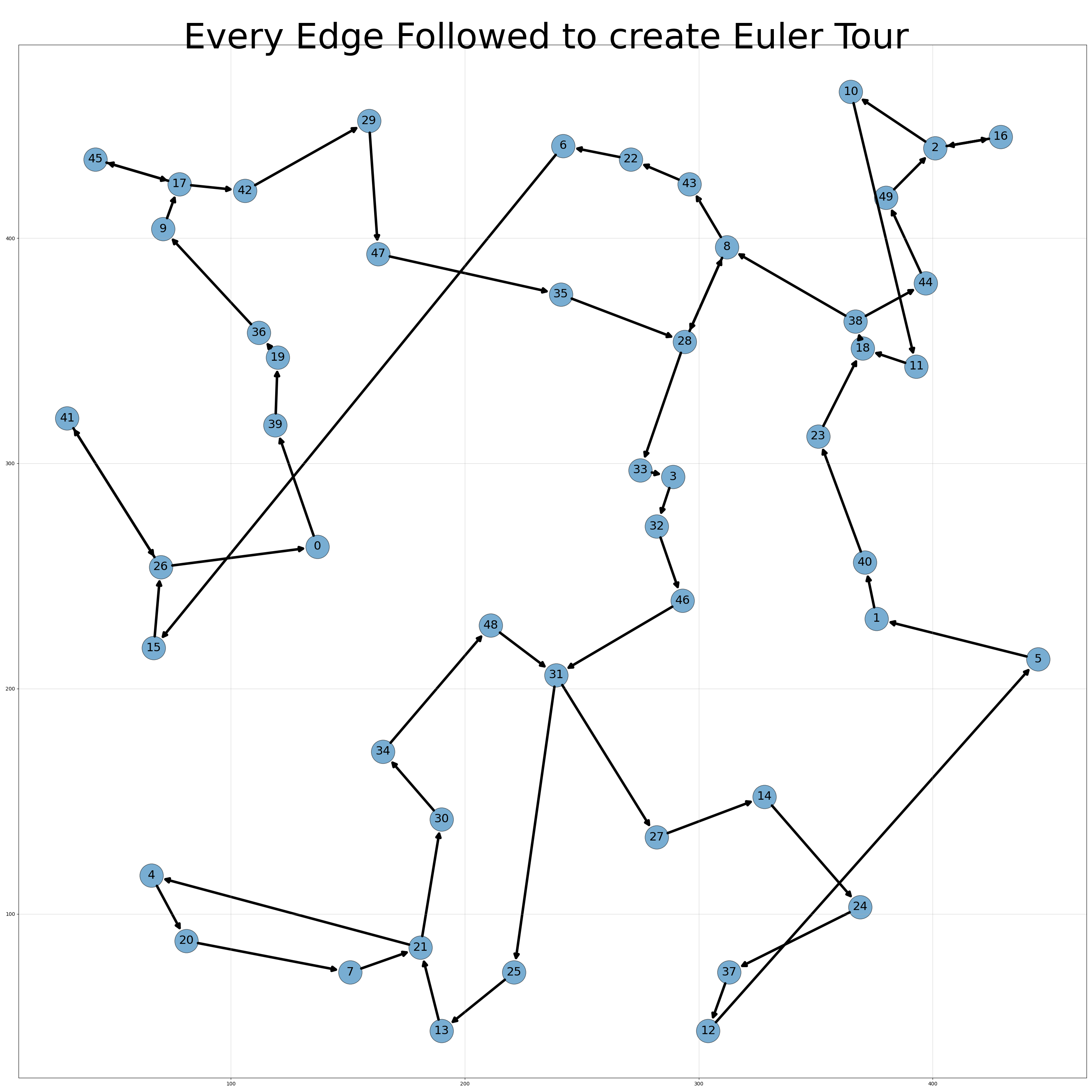
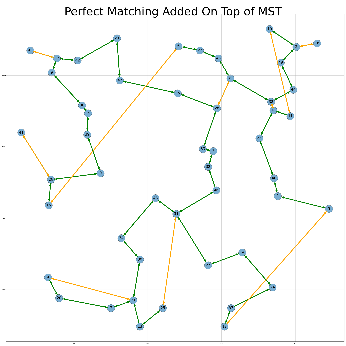
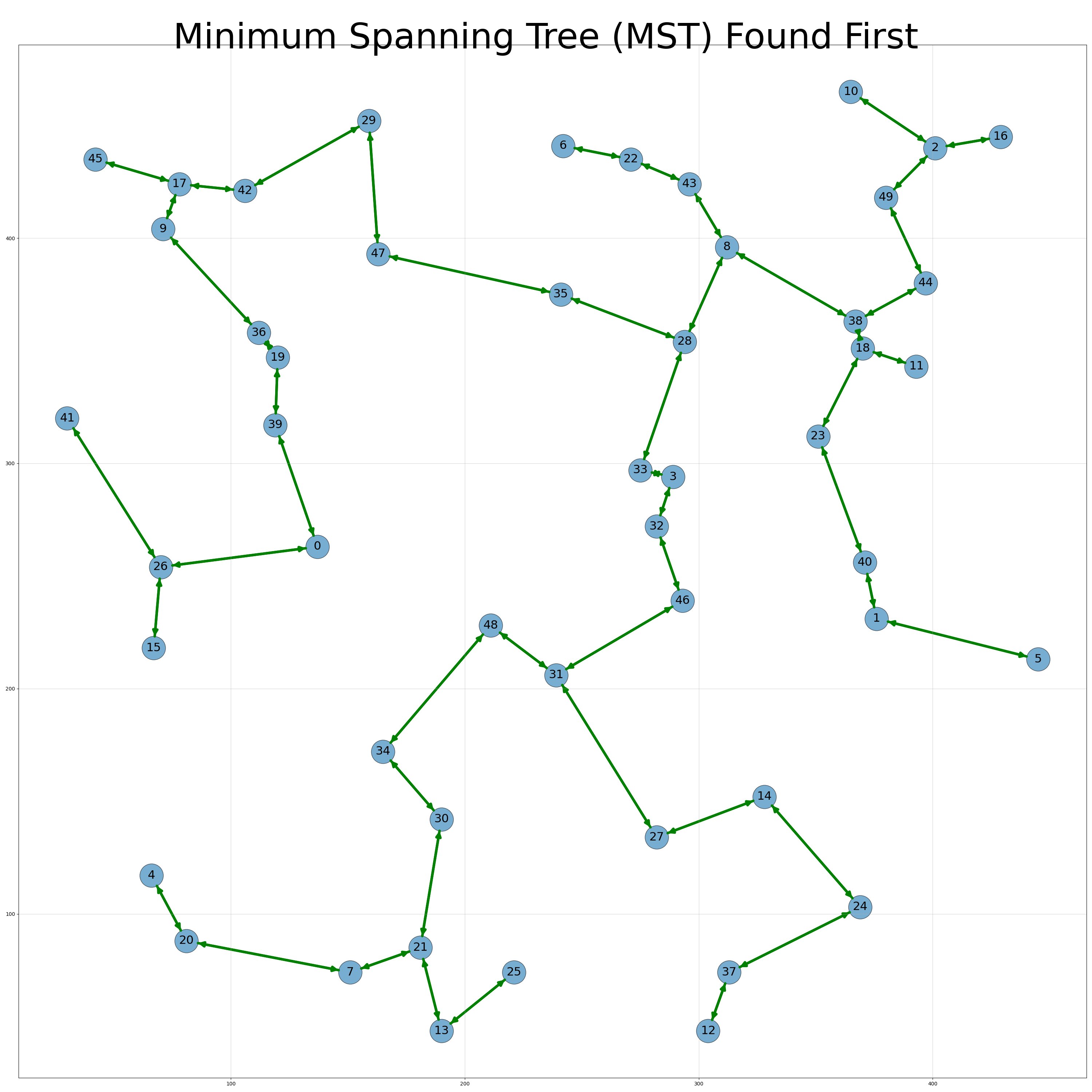
**Christofides**

The above algorithms all tend to come closer to a better solution but they provide no strong guarantees on the maximum relative tour length. After doing some research I came across the Christofides algorithm which, for an optimal tour, OPT, guarantees a tour length of no more than 1.5\*cost(OPT). It does so while running in polynomial time, thus meeting the required conditions.

**Algorithm**

The steps of the algorithm are as follows.

1. Create a maximum spanning tree (MST), T of the graph (shown in green)
2. Find the vertices with odd degree in T, let us call it O
3. Find the minimum weight perfect matching, M, from the vertices O. A perfect matching is always possible due to the handshaking lemma. (shown in orange)
4. Combine M and T to form an Eulerian Circuit (shown in black)
5. Remove repeated vertices from the combined tour to find the desired Hamilton tour. (shown in black)



**Implementation**

1. Prim’s algorithm was used to find the MST. The MST and M were represented by their edges in a 2D array. The number of edges, n, between two vertices u and v were represented by setting edges­uv and edgesvu to n as the graphs are bidirectional.
2. The sum of a row edgeu represented the degree of u and a u with an odd sum was added to a list of odd vertices. The distance matrix was sliced using the odd vertices as indexes. This new array served as input for the algorithms to find M.
3. A greedy algorithm to find M was used in the basic graph.py file for simplicity’s sake. However, by setting the default argument optimal to False when calling the Christofides function, an optimal version from the test.py file is used. The optimal version in the test.py file is an implementation of the Hungarian algorithm written by Timon Knigge. Every edge found in M was added to the total in its respective cell in the edge array.
4. To get an Eulerian tour the edge array served as a map. Starting from any row. the column with a non-zero value was chosen as the next destination in the tour. That row was then chosen as the origin for the next iteration. The last row in this case was chosen as the starting row but as it is a tour it did not make a significant difference. To cover the loops that could be left out in tour creation, the edge array was passed over and rows with positive sums were reselected as starting points for the creation of a new tour. This new tour in each iteration is added to the original tour at the position of the index at which the new loop starts.
5. The Hamiltonian tour is simply a nubbed version of the Eulerian tour. This is simply achieved by converting the perm list to a dictionary and back to a list.

**Proof of Polynomial Time Complexity.**

Each step is separate from the last in terms of time complexity for vertices, V.

1. Prim’s algorithm has time complexity O(V2).
2. As each vertex is visited once this has time complexity O(V).
3. The greedy algorithm to find M goes through the list of vertices once for every vertex in the odd vertex list and as such has time complexity O(V2). The Hungarian algorithm has slightly higher time complexity O(V3)
4. The cost of finding the Eulerian tour is the cost of traversing every edge is O(V2)
5. The cost of removing repetitions in the Eulerian tour to get the Hamiltonian tour is O(V2).

**Proof of 1.5 Performance Bound**

As the optimal tour with one edge forms a minimum spanning tree. As there can be no negative edges in the graph, cost(MST) ≤ cost(OPT). The cost of the Christofides tour can be no greater than the cost of the matchings and the MST.

Therefore, cost(Christofides) ≤ cost(MST) + cost(M) ≤ cost(OPT) + cost(M)

Any tour can be broken into two perfect matchings with the possibility of a single spare vertex for an odd number of vertices. As the matching, M is minimized, cost(M) is at least as low as the cost of the lesser of the two matchings in OPT. This is because M is essentially taking shortcuts of the tour as it uses edges between the odd vertices and the triangle equality means that any edge in M cannot have an alternative path shorter than it. As such cost(M) ≤ 0.5\*cost(OPT).

Evaluating:

cost(Christofides) ≤ cost(MST) + cost(M) ≤ cost(OPT) + cost(M) ≤ cost(OPT) + 0.5\*cost(OPT)

cost(Christofides) ≤ 1.5 cost(OPT) QED

**Experiments**

To evaluate the performance differences in the algorithms, the mean tour lengths given by the algorithms at varying numbers of nodes and densities were evaluated. The heuristics compared were the ones mentioned in this article as well as Christofides with optimal matching followed by Two-Opt. Optimal solutions were derived using an exponential algorithm. As such, for higher node quantities, calculating optimal values proved infeasible and those values are blacked out in the results table. Repeats were also done where feasible given available resources i.e., time and computational power. 5 repeats were done up to 16 nodes, 2 thereafter up to 18 and a single run thereafter. The tests were done with randomly produced Euclidean graphs, and the details of their generation and the test parameters are apparent in the test file where the supporting algorithms used are also credited. The tour lengths are shown in the table below, coloured by their relative values for each node quantity.

I hypothesized that the best performing algorithms would have their advantage heightened with a larger number of nodes, but density would have little impact on the relative performance levels. This is because while in reality higher density implies less bidirectionality in routes, in the simulated conditions all edges are completely bidirectional.

**Visualisation Analysis**

From the visualization, it can be seen that the swap heuristic followed by the simple greedy algorithms are the worst performing for almost all cases. Furthermore, at higher node quantities the relative performance of the swap heuristic worsens.

The Christofides algorithm with optimal matching performs slightly better than the aforementioned, especially as the number of nodes increases. The two opt algorithm when preceded by the swap heuristic performs very well in all scenarios but is consistently outperformed when preceded by the Christofides algorithm with optimal matching. This discrepancy is exacerbated as the number of nodes increases. Additionally, at both high and low quantities of nodes, no conclusive pattern could be seen in the relative performance levels of the algorithms when density was varied.

Additionally, in the scenarios tested, the optimal Christofides algorithm never exceeded its promised bound of 1.5 times the optimal tour length where the tour length was available. Furthermore, the optimal Christofides algorithm never returned a length more than 13% higher than the best length found by any other algorithm. However, this pales in comparison to the two-opt heuristic which strays above 7% above the best tour length when preceded by the swap-heuristic. Even then, the tour length is only 17 units longer as this occurs at a low node count at just 6 nodes.

When the optimal Christofides and two-opt are combined, the tour length never exceeds 2.7%. Although this was predicted to be the best performing, the observed bound on performance is impressive.

**Conclusion**

This analysis demonstrates that while the Christofides algorithm provides a well-performing backbone through its minimum spanning tree, the Christofides algorithm suffers from performance loss in the matching phase. The two-opt heuristic performs well in the area that the Christofides algorithm has its shortcomings. The effects of the additional Christofides algorithm only offers minimal tour length benefits when used in tandem with the two-opt heuristic compared to the two-opt heuristic on its own. As such while it can be noted that a benefit is found with the Christofides algorithm, the two-opt heuristic appears to be by far the best performing algorithm in head-to-head comparisons. These conclusions are mostly in line with the hypothesis. However, the excellence of the two-opt heuristic was not predicted and should be considered in further experiments.

It should also be noted that none of the algorithms besides the swap heuristic on its own routinely went close to exceeding the best route length by more than 50% despite only the Christofides algorithm promising this

**Possible Extensions**

The advantages two-opt provides in Euclidean graphs is useful as the scenario is a common enough one. However, the performance of the heuristic in other conditions should be investigated as the benefits of “uncrossing” the path can be mitigated. These non-Euclidean based experiments can be analysed not just in the case of networks but of focused scenarios, such as intra-city road networks vs inter-city ones as one would expect a truer reflection of the effects of density in such scenarios. The algorithms could also be tested to limits where it is more challenging to come closer to an optimal solution, providing a more testing analysis.