IADS cw-3 Report

Part C - Your Own Algorithm

For this section I have implemented the farthest insertion algorithm. This works quite differently from other travelling salesman algorithms as it starts my finding the furthest away nodes. Then it repeatedly finds the city (one that is not already in the tour) that is the furthest away from any city in the tour and inserts it in between whichever cities would result in the tour being as short as possible.

However, this does not find the optimal solution for all cases as it is a construction problem and often this algorithm can be improved by adding an optimizer.

The farthest insertion algorithm has 3 key stages:

1. **Initialisation**
    - Finding the nodes furthest away from each other.
    - Start a sub-graph containing this maximal journey in the form of (i-j-i)
2. **Selection**
    - using the given sub-tour, find node (in this case r) that is not already in subtour that is furthest away from any node in the subtour.
3. **Insertion –**
    - Find which city in the subtour r is closest to and insert it next to this city and create a new edge.
    - Repeat step 2 and 3 until all nodes have been added to the tour.

I found this algorithm in a paper which discussed many different heuristic approaches to the travelling salesman problem. This paper is referenced below.

“11 Animated Algorithms for the Traveling Salesman Problem” by Lawrence Weru.

https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/

The time complexity for this algorithm is O(n^2) because for each node selection it performs an insertion sort to find the optical location for that node in the tour.

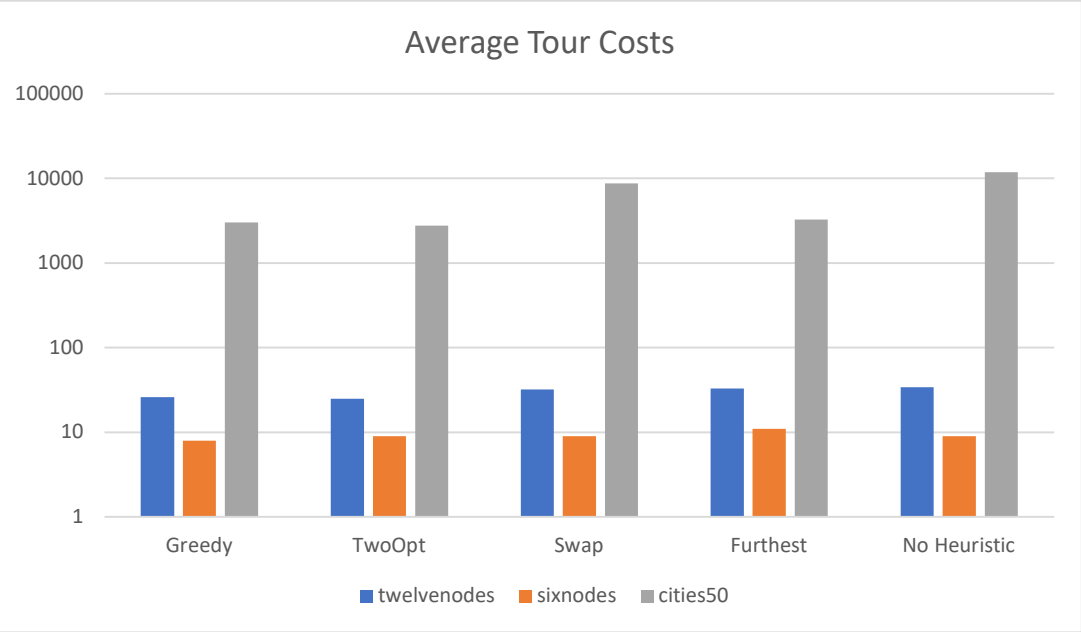
Part D – Experiments

For this section I created two methods that will create random graphs - NonMetric(nodes, limit) and XYNodes(n, Xlimit, Ylimit).

The first experiment I did looked at which algorithms provided the best tour value for the data I was given. For each Heuristic used in the previous parts including the Farthest Insertion Sort Algorithm, I ran each heuristic through 100 iterations using the datasets provided. To provide a control I also ran each dataset with no heuristic.

Below is a table and graph showing my results from this experiment.

	twelvenodes.txt	sixnodes.txt	cities50.txt
Greedy	26	8	3011.593191973481
TwoOpt	25	9	2781.2725822565517
Swap	32	9	8707.056392932434
Furthest	33	11	3726.256910080557
No Heuristic	34	9	11842.557992162185

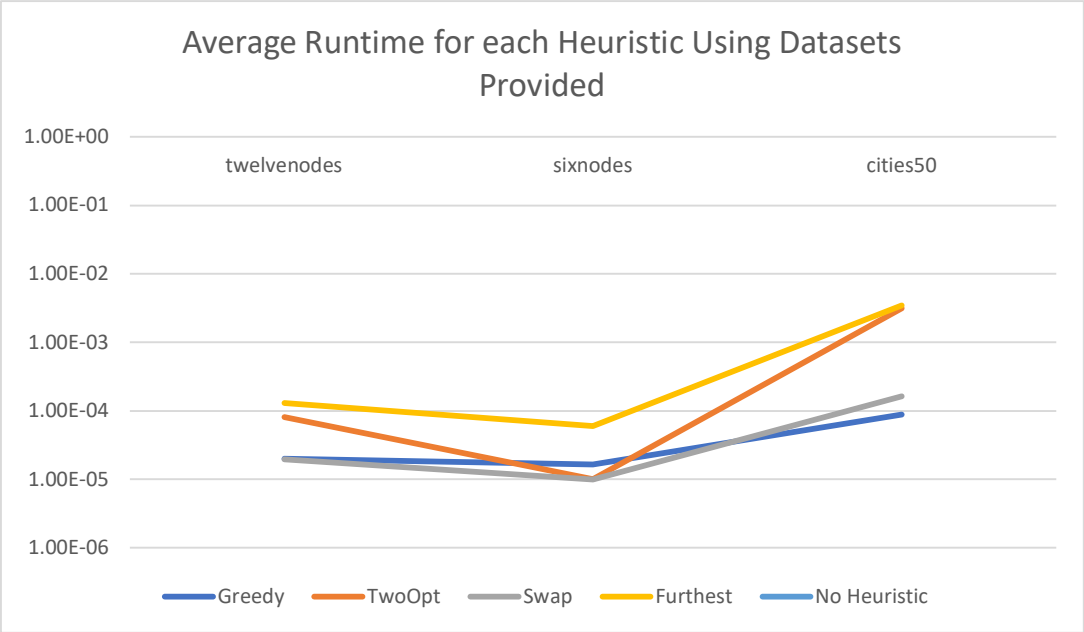


From this we can conclude that for a small number of nodes (e.g. sixnodes.txt and twelvenodes.txt) and for larger number of nodes in the Euclidean form, that in general Two Opt heuristic performs the best at providing the optimal solution.

As we can also see the Furthest Insertion algorithm I created doesn't provide the optimal solution for any of the provided datasets but it does provide a better tour value for cities50 than swap but not any of the other ones, this shows this heuristic works best for Euclidean.

The next experiment I looked at was finding the runtimes for each heuristic on each of the datasets provided. This allowed me to compare how each algorithm performed as each data set got larger and to see which algorithms run the fastest i.e has the best performance. I ran each heuristic algorithm 100 times , collected the time taken for each of these 100 iterations and then calculated an average time taken. Below is a table and graph showing my results.

	twelvenodes.txt	sixnodes.txt	cities50.txt
Greedy	2.0003318786621095e-05	1.652717590332031e-05	8.852481842041016e-05
TwoOpt	8.059024810791015e-05	9.9945068359375e-06	0.0031507062911987306
Swap	1.9636154174804687e-05	9.968280792236329e-06	0.00016357421875
Furthest	0.00012939929962158203	6.006956100463867e-05	0.0034733033180236817
No Heuristic	0.0	0.0	0.0



As we can see from this graph the less nodes we have the shortest time the heuristic will take to run and as we in increase the number of nodes for each heuristic. We can also see that is the greedy algorithm and Swap algorithm that seem to have the best run times for each of the provided datasets. This is interesting because swap had the worst value for cities50, so although swap doest provide many optimal solutions it makes up for in runtime. It is also clear that based on run time the algorithm I created (farthest insertion) has the worst performance but is similar to Two Opt for cities 50.

I then also looked at how the algorithms performed on the random graphs I had generated and how varying the “window” of the plane to take in larger/wider values for the x and y co-ordinates affected the tour values. For this I created the algorithm XY nodes which creates datasets similar to cities50 but I can choose the number of nodes and choose the limit of the x values and the y values.

To experiment with this I varied the number of nodes and the x and y limits. I chose 3 different values 20,40 and 80 and for each of these I tested with the same x and y limits -50,400,1000 so that I would easily be able to see the correlation between my results. To test this I used the same testing algorithm I had used before (that iterated through it 100 times) and tested each of these new graphs. The results are shown in a table below.

Tour Values for Different Cities

	cities20			cities40			cities80		
Algs,Limits	50,50	400,400	1000,1000	50,50	400,400	1000,1000	50,50	400,400	1000,1000
Greedy	223	2046	4033	254	2259	6526	426	3537	8750
TwoOpt	181	1532	3448	243	2047	5699	383	2999	7747
Swap	373	3827	6938	799	6240	14451	1578	11795	30664
Furthest	251	2642	5398	389	3156	8368	603	4009	11330
No Heuristic	547	4059	9617	960	8842	18980	2094	15670	41562

From this table we can see the impact of increasing the number of nodes and the impact of widening the x and y coordinates on the tour values. We can conclude that by increasing the number of nodes but keeping the limits the same increases the tour value and increasing the limits but keeping the number of nodes the same also increases the tour value.

We can also see that swap heuristic has the worst performance for each city.

By running this experiment can also conclude that by increasing the number of nodes and widening the range of coordinates it also increases the runtime.