Orlagh Keane S2084384

IADS CWK3 – Report

Part C – Algorithm

I have implemented a heuristic called Furthest Insertion Sort. This algorithm includes three steps, firstly initializing the tour, secondly, selecting a furthest node and then lastly inserting it in the optimal position in the tour.

These are described in more detail below.

1. It begins with a random city, then connects it with the furthest city from it. A subtour is created which contains the beginning node, the furthest node from it, and then the beginning node.
2. Next, it finds a node which is not in the tour which is furthest from any node already in the tour.
3. It finds it best placement by comparing it with every possible location in the tour. It finds that the best placement will be in between two nodes which causes the least increase in length. For example, if the two nodes next to each other in the tour were **i** and **k**, and the city being inserted between them was **j**, then the algorithm would minimize the calculation **d(i, j) + d(j, k) - d(i, k).**

This will be repeated until every city has been included on the tour. Once all the nodes have been added to the tour it ends and the tour value is calculated.

The algorithm has a polynomial runtime of $O(n^2)$ as for each node selection there is an insertion sort to find its optimal location in the tour.

After doing some research I decided to write the furthest insertion sort. The paper below detailed the nearest insertion sort and explained the similarity and differences of it to the furthest insertion sort and so I decided to implement this heuristic.

Rosenkrantz, D.J., Stearns, R.E. and Lewis, II, P.M., 1977. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, *6*(3), pp.563-581.

Orlagh Keane - S2084384

Part D – Experiments

For each of the three sets of data provided , 'sixnodes.txt', 'twelvenodes.txt' and 'cities50.txt', I carried out each of the algorithms Swap Heuristic, Two-Opt Heuristic, Greedy Algorithm and the Furthest Insertion Algorithm. Since the Furthest Insertion Algorithm works best for Euclidean entries, so I decided I would test similar files to "cities50", such as "cititesː75" which I created. The tables of results are presented below. Firstly, for each the datasets, and then after for the average tour value and the average time taken, based on iterations of 50. I included the values when no algorithm had been implemented, as a control.

SixNode Results

Algorithm	Average Tour Value	Average Time Taken
No algorithm	9.0	0.0
Swap	9.0	4.3382644653320314e-05
2-Opt Heuristic	9.0	2.7718544006347657e-05
Greedy	8.0	2.0036697387695314e-05
Furthest Insertion	11.82	5.380630493164063e-05

TwelveNode Results

Algorithm	Average Tour Value	Average Time Taken
No algorithm	34.0	0.0
Swap	32.0	0.00010056495666503907
2-Opt Heuristic	34.0	0.0002917003631591797
Greedy	26.0	4.041671752929688e-05
Furthest Insertion	37.12	0.0001531982421875

Cities50 Results

Algorithm	Average Tour Value	Average Time Taken
No algorithm	11842.557992162185	0.0
Swap	8707.056392932442	0.0003096675872802734
2-Opt Heuristic	11842.557992162185	0.007985072135925293
Greedy	3011.593191973484	0.0007368135452270508
Furthest Insertion	10817.028966535481	0.001773686408996582

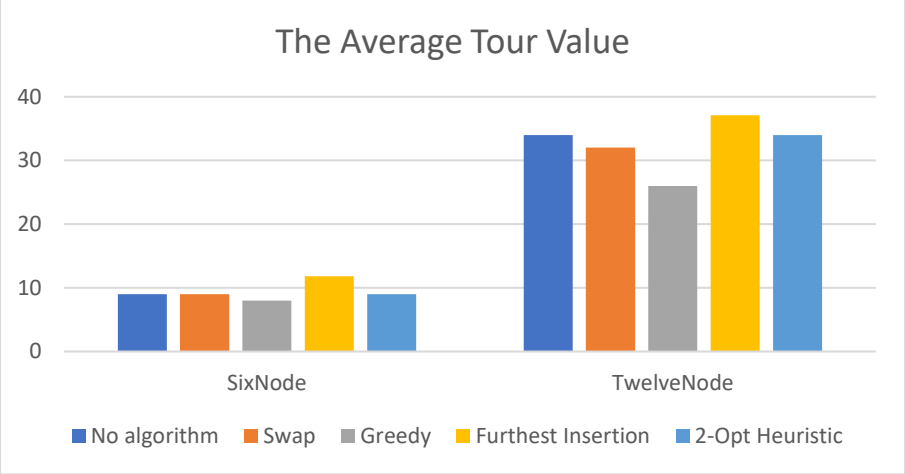
Cities75 Results

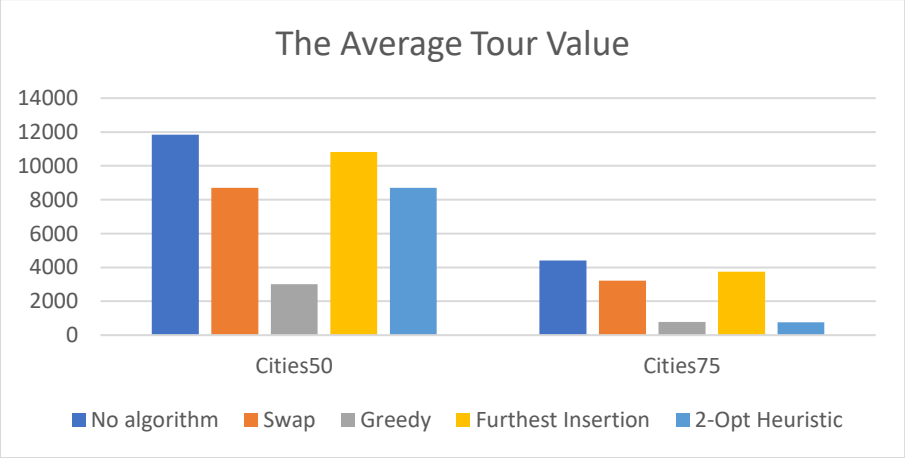
Algorithm	Average Tour Value	Average Time Taken
No algorithm	4400.867985777469	0.0
Swap	3219.2383618105373	0.0006737804412841797
2-Opt Heuristic	767.3812013977735	0.013133502006530762
Greedy	775.6879229005718	0.001137847900390625
Furthest Insertion	3740.891411722929	0.002194657325744629

Orlagh Keane - S2084384

Average Tour Value

	No algorithm	Swap	2-Opt Heuristic	Greedy	Furthest Insertion
SixNode	9.0	9.0	9.0	8.0	11.82
TwelveNode	34.0	32.0	34.0	26.0	37.12
Cities50	11842.557992162185	8707.056392932442	8707.056392932442	3011.593191973484	10817.028966535481
Cities75	4400.867985777469	3219.2383618105373	767.3812013977735	775.6879229005718	3740.891411722929





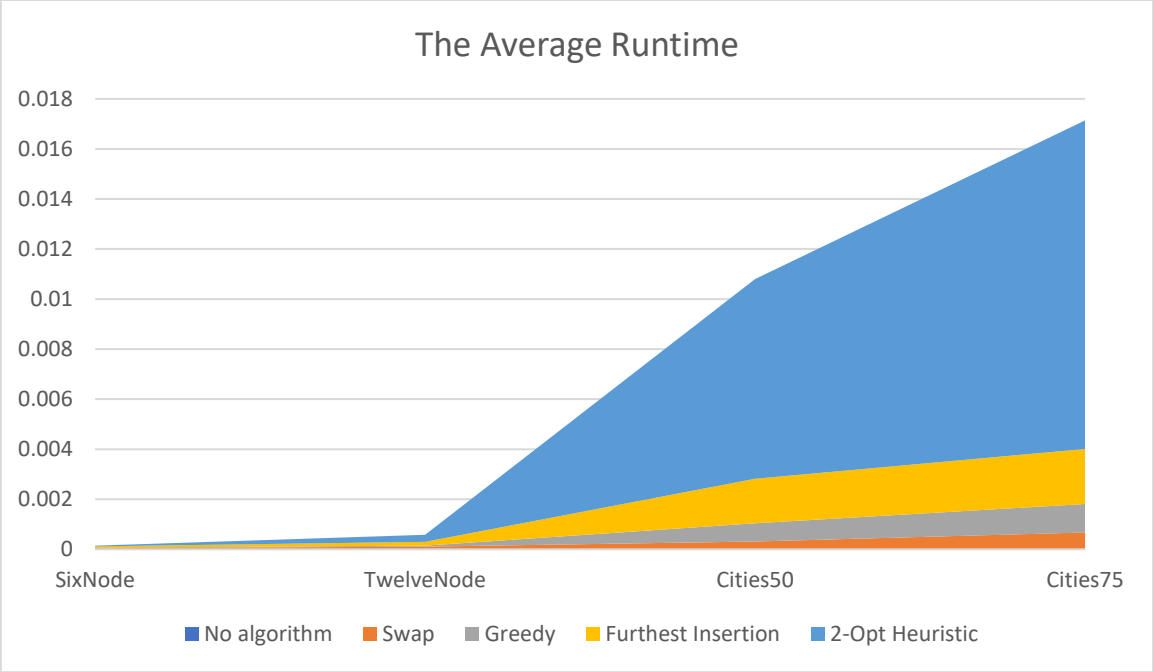
From both the graphs above, both represented on different scales as the values are significantly different, it is clear that the furthest insertion sort has a bigger tour value than the other heuristics implemented. However, it is notable that it makes an improvement to the cities50 and cities75, which may suggest that the algorithm performs better for Euclidean entries.

Both the heuristics and greedy make improvements to the tour value, when compared with the control value. Interestingly, the 2-Opt heuristic makes no improvement to the control for six and twelve nodes, however it improves the value significantly for cities 50 and 75. It actually produces the best tour value for cities75 and therefore you would expect it to continue to improve for bigger sets of Euclidean values.

Orlagh Keane - S2084384

Average Time Taken

	No algorithm	Swap	2-Opt Heuristic	Greedy	Furthest Insertion
SixNode	0.0	4.3382644653320314e-05	2.7718544006347657e-05	2.0036697387695314e-05	5.380630493164063e-05
Twelve Node	0.0	0.00010056495666503907	0.0002917003631591797	4.041671752929688e-05	0.0001531982421875
Cities50	0.0	0.0003096675872802734	0.007985072135925293	0.0007368135452270508	0.001773686408996582
Cities75	0.0	0.0006737804412841797	0.013133502006530762	0.0011378479003906 25	0.002194657325744629



To see if the algorithms created any improvements, I decided to record the average runtime of each algorithm on each dataset. The results of this can be seen in the table above, and the results are represented on the above graph. It is clear that the 2-Opt Heuristic has the longest runtime. It has slightly better tour value than the furthest insertion sort however the runtime is significantly better.

While there is not one algorithm which is clearly the best overall, we can see that the Greedy Algorithm has the lowest average cost tour for all but “cities50”. While the swap algorithm has the lowest runtime cost, it has much higher values for average tour cost. The greedy algorithms runtime cost is closely second to that of the swap, and therefore I would suggest that it is the optimal algorithm.

Next I decided to explore the effect of varying the range of values from which the Euclidean data is created from. After adjusting the Euclidean setting for “cities75” so that it created values with a larger range of values, there was an overall increase in the average tour values however the runtime does not vary much and there is no trend. This makes sense as the run time is mostly influenced by n, which is 75 in each case and therefore should be similar. I used the range for x and y as (75, 75), (500, 500) and (2000, 2000).

Average Tour Value			
	75, 75	500, 500	2000, 2000
No algorithm	3112.7608490521843	18165.651425803168	80552.90937240876
Swap	2562.047967922816	14778.45926256404	63508.659326503184
2-Opt Heuristic	541.7996535367265	3687.225670613717	14896.491093501067
Greedy	690.082233065274	4205.437115720821	15597.223751537533
Furthest Insertion	2941.2403060614515	18323.45699893176	78331.99497135355

Average Runtime			
	75, 75	500, 500	2000, 2000
No algorithm	0.0	0.0	0.0
Swap	0.0003213024139404297	0.00038188934326171875	0.0003558731079101563
2-Opt Heuristic	0.013078269958496093	0.011723098754882812	0.009119143486022949
Greedy	0.0007199668884277344	0.0007007980346679688	0.0006924867630004883
Furthest Insertion	0.0013744688034057616	0.0017548179626464843	0.001339268684387207