Introduction to Algorithms and Data-Structures

Coursework 3

Heuristics for the Travelling Salesman Problem

By:

Divy Bramhecha

S1885517

Part C:

The travelling salesman problem is quite interesting and is known to be NP-Complete[1]. Solving it will lead to better result in a lot of its applications. I am not going to attempt it here to solve a general polynomial time algorithm, as that is beyond my skills now. However, there are many versions of the problem, on which specifically we can design algorithms that might give better and faster results. One such problem that interests me is in video game crawls[2]. They typically involve 3d structures and Euclidean distances. It's quite likely that you wouldn't be able to travel in straight lines, but it's certain that the environment will follow the triangle inequality and would also be symmetric. To tackle this, I did some research and found Christofide's algorithm to be quite interesting which matched the above requirements. It claims to guarantee solution withing 1.5 error[3].

The Christofide's Algorithm:

It can be described in few simple steps:

1) Find the minimum spanning tree T of the graph.
2) Find the Odd degree of vertices O. (They are even in number due to handshaking lemma.)
3) Find the minimum weight perfect matching M from the vertices O.
4) Combine M and T to form a Eulerian circuit.
5) Now form a Hamiltonian circuit from the circuit in the previous step.

My implementation of the algorithm follows the above steps correctly. The following are the running times for each of the steps.

1) $O(V^2)$, I used Prim's algorithm designed for adjacency matrix. Note, it can be improved to O(E.log(V) if using heaps.
2) O(V) Goes through all the vertices once.
3) $O(V^2)$. Note, this does not find the perfect minimum weight matching. I decided to implement a greedy solution which gives an approximation of the perfect minimum weight matching.
4) $O(V^2)$. This just traverses the combined graph of M and T.
5) O(V). This removes any extra edges occurred.

As we can see, the algorithm consists of combining steps 1-5, and so it runs in $O(V^2)$ time, which is polynomial. I have annotated the algorithm in the implementation properly for you to verify it.

Next, we'll verify why Christofide's algorithm is said to be a 1.5 approximation for metric cases.

The weight of a minimum spanning tree is less than the optimal solution:

1) Take an optimal tour of cost OPT (optimal solution).
2) Drop an edge to obtain a tree T.
3) All distances are positive so cost(T) <= cost OPT, Hence cost(MST) <= cost OPT.

---

[1] It's the decision version of the problem that is NP-Complete.

[2] The fastest way to complete a given game from start to finish, often involves visiting the same cities multiple times for open ended quests in various orders flexible to the players choice. A lot of youtuber's do this.

[3] Before that I also implemented the 2-Approximation algorithm.

The two-approximation algorithm traverses the Minimum spanning tree twice, at most repeating all vertexes V, 2V times. Resulting in the cost to be doubled. Hence:

$$cost(MST) <= OPT <= 2.cost(MST).$$

We improve this algorithm, by instead of traversing all the edge two times, we can make shortcuts between odd vertices. Therefore, the cost of the tour becomes:

$$cost(MST) + cost(Perfect\ Matching\ PM)$$

$$cost(MST) + cost(PM) \leq cost(OPT) + cost(PM)$$

Bounding, PM.

1) Take an optimal tour of cost OPT.
2) Consider the odd vertices O.
3) Shortcut the path to only use O.
4) As they have perfect matchings, they partition into M_1 and M_2, such that cost(M1)+Cost(M2) <=OPT.
5) Therefore Cost(PM) <=OPT/2.

$$cost(MST) + cost(PM) \leq cost(OPT) + \frac{1}{2}cost(OPT)$$

Therefore, Christofide's Algorithm is said to have a 1.5 approximation bound for TSP-metric problems.

Testing:

The straightforward way to compare algorithms would be to:

$$\alpha = \frac{cost(Found\ Solution)}{cost(Optimal\ Solution)}$$

However, finding the optimal solution of any graph is itself NP-Hard. However, we could instead compare it with a lower bound on the optimal solution. As we saw in the previous section that the weight of the minimum spanning tree is at most optimal solution:

$$w(MST) \leq cost(Optimal\ Solution)$$

Hence, while comparing algorithms we will use the cost of MST to compare the solution, i.e.:

$$\alpha = \frac{cost(Found\ Solution)}{cost(MST)}$$

First testing on the provided tests:

	Two Swap	Two Opt	Greedy	2-Approx	Christofide's	Optimal[1]
Cities25	5027	2233	2587	2456	2509	1636
Cities50	8707	2686	3011	3477	2844	2204
Cities75	13126	3291	3412	4104	3592	2607

[4]

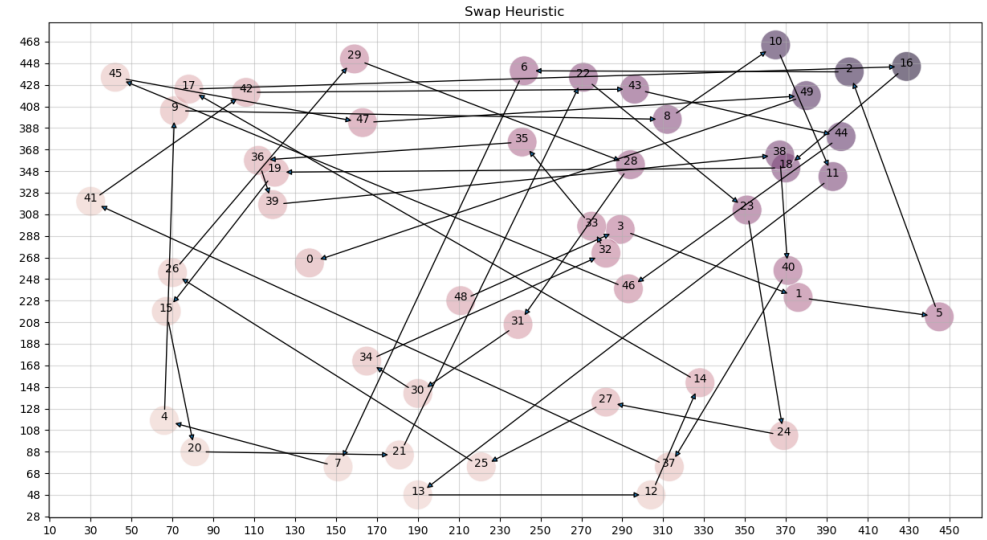
Also, please scroll down to see how the solutions look like, I used them to understand what the algorithms were doing better.

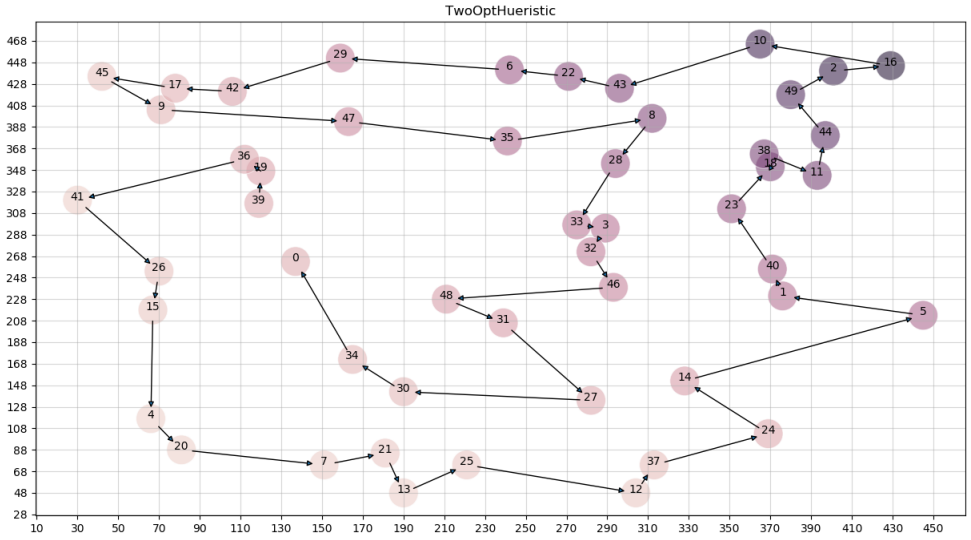
The Two Opt Heuristic seems to outperform all the heuristic in the above graphs. Certainly, we could design graphs which are more clearly favouring algorithms like the Greedy approach, for example where we design a circle with equidistant points. Regardless, first I start by generating lot of random graphs of a lot of different testing sizes and analyse those results.

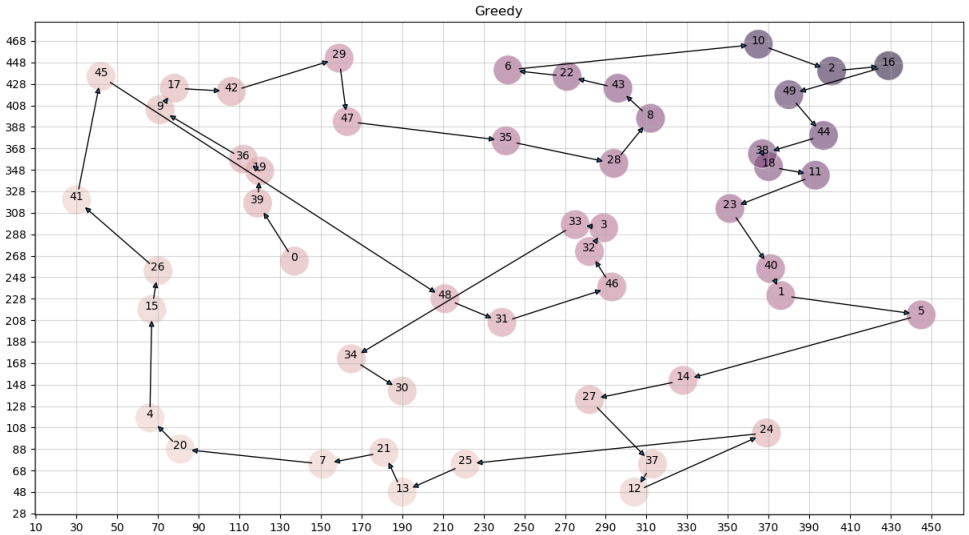
Please refer answers.csv for more detail on the random tests done.

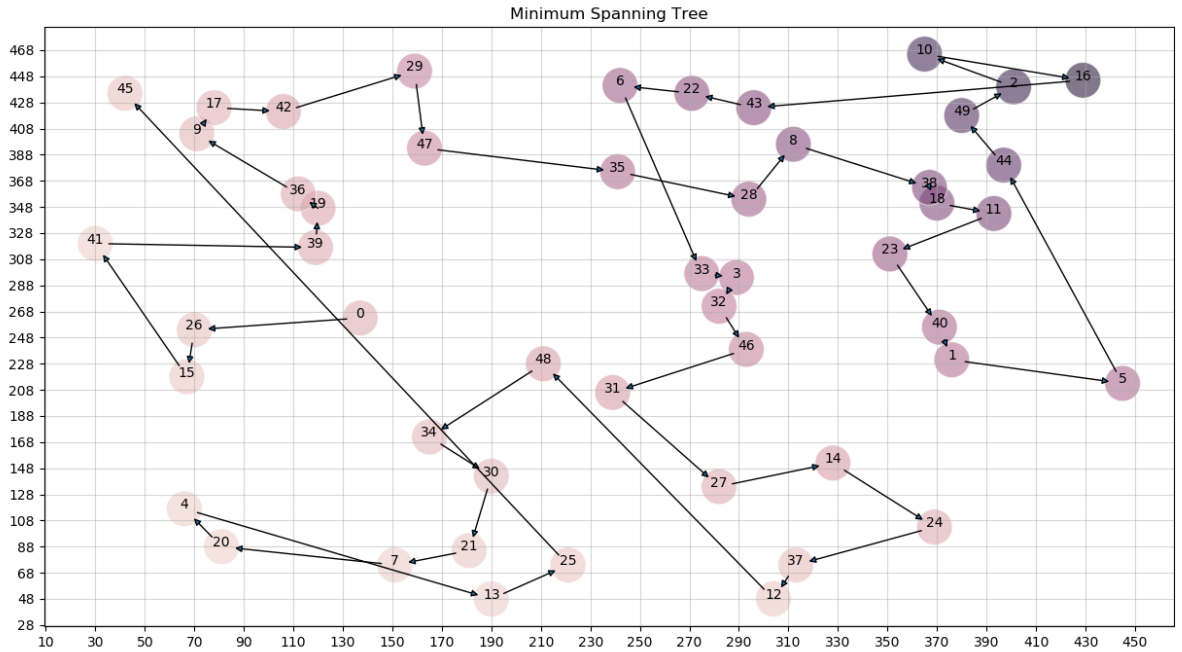
[4] Please note that this optimal cost only refers to the weight of the minimum spanning tree as described above.

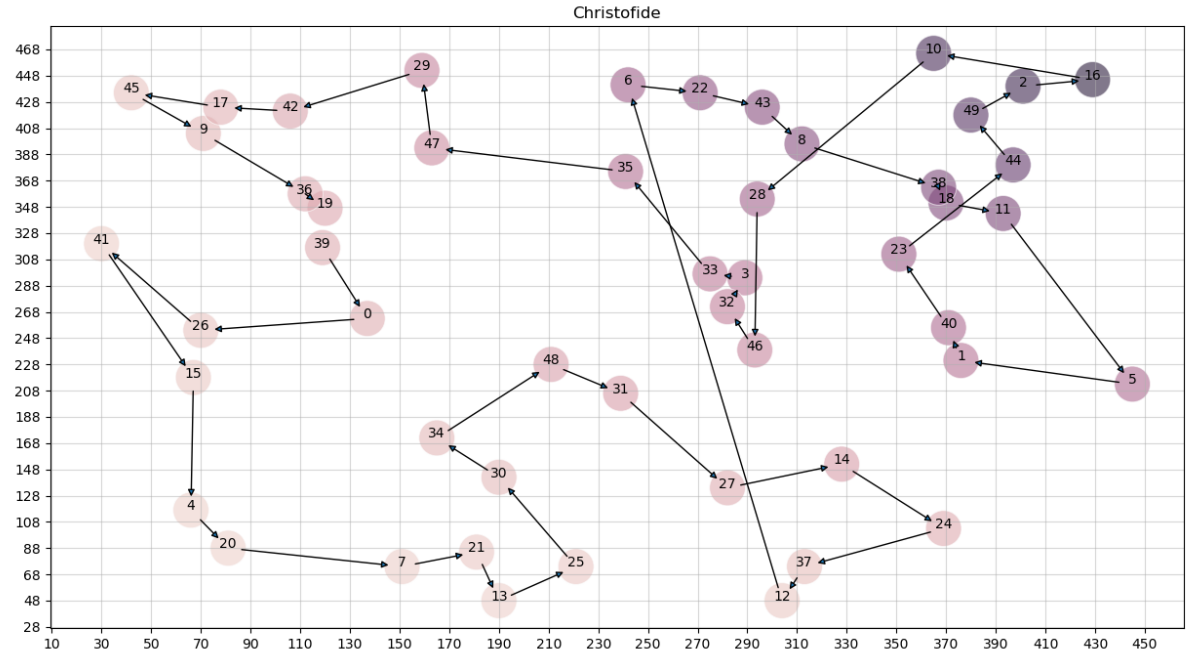
All solutions on cities 50 graph.








Minimum Spanning Tree


Christofide