Sultan Alanazi

Miles McCall

Austin Sanders

CS 520/420 Final Project

Travelling Salesman Problem

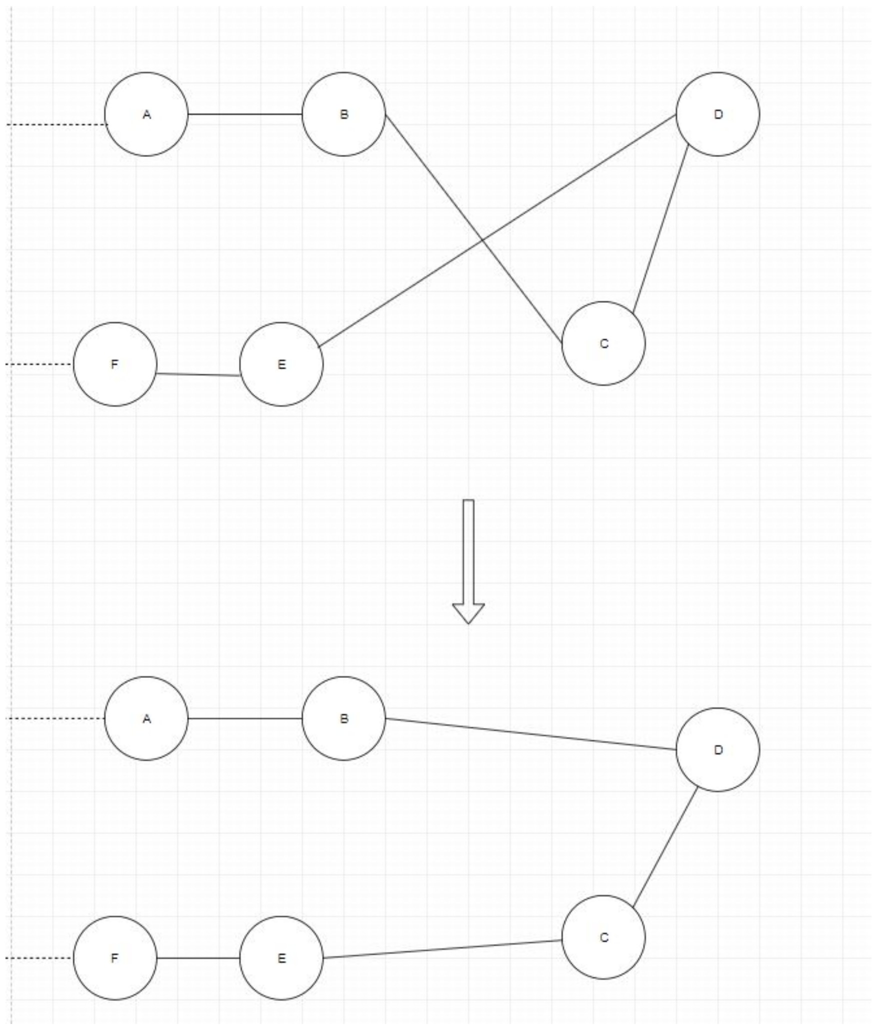
A description of at least three different methods/algorithms for solving the Traveling Salesman Problem.

Nearest neighbor: The nearest neighbor algorithm is a greedy algorithm and one of the easiest solutions to implement for the traveling salesman problem. You simply pick any vertex as the starting point, in the scope of our problem this would be the starting city. The algorithm would then choose the shortest outgoing edge from this vertex. The algorithm would then change the vertex from the starting position to the newly chosen vertex, and mark it as visited. From here the algorithm will then analyze all of the outgoing edges and pick the shortest one that has not been visited. This algorithm will execute the above steps until every vertex has been visited then execution will stop. The order of the vertex chosen will then be the solution to the traveling salesman problem with the nearest neighbor algorithm.

This algorithm of course has some drawbacks. Due to the greedy nature of the algorithm it is entirely possible, and common, for the algorithm to choose a slower path over a shorter path that can be found from basic human reasoning. The benefits of this algorithm are the fact that is simple and easy to write. The drawbacks are that it is not always accurate.

The running time of this algorithm is O(V^2) where V is the number of vertices in the given graph.

2-opt: 2-opt is an optimization algorithm used in solving the traveling salesman program. The core idea behind this algorithm is to take two routes that previously crossed over each other, and reorder the graph in a way where they do not anymore. I have created two sample graphs to illustrate what I mean when I say this:



Of course the real algorithm done on a real graph would run many more times rearranging the edges as many times as it could until an optimized version of the graph is found. This algorithm will keep swapping edges until every single permutation has been tried, and no more improvements can be found. The obvious drawbacks to this algorithm are that it is very expensive on resources, is difficult to implement, and takes a lot of time since it has to try every combination. The benefits are that the algorithm will find an optimized path for the salesman to take which will be way more accurate than the previous greedy algorithm.

The time complexity for the 2-opt algorithm is $2^{O(n)}$. A very expensive algorithm that only gets exponentially worse. It is feasible for graphs that contain a small amount of vertice and edges but becomes increasingly difficult the more you add to it. Escalating to NASA level supercomputers needed to compute it in a reasonable time.

Simulated annealing: Simulated Annealing is a probabilistic technique for approximating a global optimum. The name comes from annealing in metallurgy, a technique where the smith cools and heats the metal in a controlled process in an attempt to increase the size of the crystals and reduce its defects. This analogy comes into play with simulated because sometimes the algorithm will accept a worse tour in an attempt to get out of a local minimum in pursuit of finding the global minimum.

The first solution is chosen via the nearest neighbor method. Then we keep running the the solution until we reach a local minimum (reached a sufficiently cooled temperature). We then select a different neighbor with a change to the graph, and run the solutions again to see if we get a better result with a lowered temperature (tour length). We run this algorithm over and over again until no improvements can be made. The key to this solution is sometimes we will accept a longer tour length with the goal of escaping a local minimum for the sake of finding the global minimum.

This is an extremely costly method of solving the traveling salesman problem. It isn’t even really a solution itself, but a statistical model in which to analyze the route in a way that produces the best solution. The worst case running time becomes exponential depending on how many edges and vertices we have. The run time T_n of simulated annealing has the complexity: $T_n = O(\,(n^2 + n) * log(n)\,)$

Pseudocode for the two algorithms that you implemented.

The two algorithms we used are built on the nearest neighbor algorithm, with both 2-opt and simulated annealing generating improvements to the initial tour.

2-opt:

```
1. Apply the Nearest-Neighbor algorithm as an initial solution.
2. Loop through the tour comparing every valid combination of the swapping function
   2optSwap(route, i, k)
       1. take route[0] to route[i-1] and add them in order to new_route
       2. take route[i] to route[k] and add them in reverse order to new_route
       3. take route[k+1] to end and add them in order to new_route
       return new_route;
```

Simulated annealing:

```
1. Apply the Nearest-Neighbor algorithm as an initial solution.
3. Loop until…
       a. The system has sufficiently cooled down
       b. A good approximate solution has been found
4. Select a neighbor solution by making a small change to the current solution
5. Decide whether to move to the neighbor solution
6. Reduce the temperature value and continue the loop
```

A discussion on why you selected the algorithm(s).

We chose the two algorithms because they provide different performance when solving the travelling salesman problem due to their unique characteristics. Our 2-opt algorithm is optimized to run on smaller graphs as it is closer to a brute force technique. It will generate efficient tours at the cost of longer computation time. Simulated annealing performs better on larger graphs and was used to generate results when the 2-opt solution wouldn't finish in time.

Your best solution for each example instance.

Examples	Tour Length	Time (sec)
1	120362	156
2	3120	416
3	1926601	974

Your best solution for each test instance.

Test Cases	Tour Length
1	5637
2	8470
3	14756
4	19472
5	26456
6	38619
7	65317