**Abstract**

The Traveling Salesman Problem has been solved relentlessly through numerous different methods. A simple and quick solution is solving via a greedy algorithm. This paper shows the strengths and faults of the greedy algorithm and attempts to improve upon it by a custom algorithm that maintains the significant speed advantage, while focusing on improvements in optimization.

**1 Introduction**

The following sections will describe our teams efforts in understanding the fundamental concepts of the Traveling Salesman Problem and describe our basic greedy algorithm. More detail will then be provided into how we were able to improve upon the greedy algorithm. Data describing these improvements will be provided along with more analysis.

**2 Greedy Algorithm Solution**

**2.1 Implementation**

Our greedy algorithm starts at the first city in the supplied list of cities. From there it attempts to iteratively connect the next closest city. This continues until the original city is reached again or a connection could not be made. The algorithm is them performed again on the next city in the original list. The best solution so far (BSSF) is observed and then recorded when the algorithm finishes on all cities.

**2.2 Big-O Complexity**

Each iteration of the greedy algorithm takes O(n) because it loops through every city in the list. Since the algorithm then runs again on every possible starting point, the final complexity is O(n2).

**3 Improved Greedy Algorithm**

**3.1 Implementation**

The improved algorithm selects a starting city and then determines the closest two cities. These are then connected to create a starting triangular path. The algorithm then selects the next closest city to two points in the starting path, connects them, and removes the path from the original two points. This continues until all possible cities are added to the list. The algorithm is then run on all possible starting points to determine the best solution.

**3.2 Big-O Complexity**

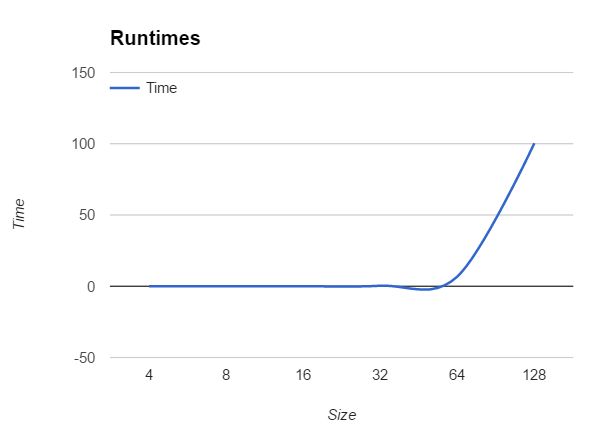
The big-O complexity of different segments of the algorithm are as follows:

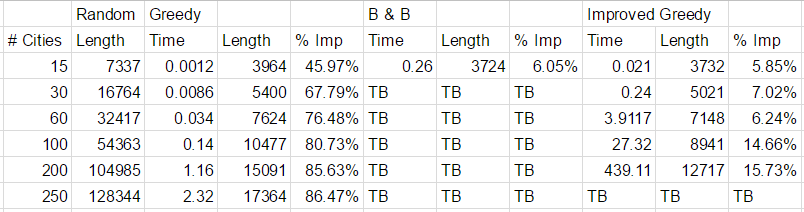
1. Clone input: O(n)
   1. Find initial triangle: O(n)
   2. Add on all other cities: O(n)\*O(j)\*O(k)   
      = O(n)\*O(j)\*O(n-j)  
      ≈ O(n)\*O(nlog n)  
      = O(n2log n)
      1. While there is a free node: O(n)
         1. Cycle through all free points: O(j)
         2. Find distance from free point to hull: O(k)
   3. Return result: O(c)

The overall time complexity then becomes O(n2log n) for each iteration of the algorithm.

**3.3 Runtimes**

The following graph shows the real values of the algorithms runtimes for various problem sizes.



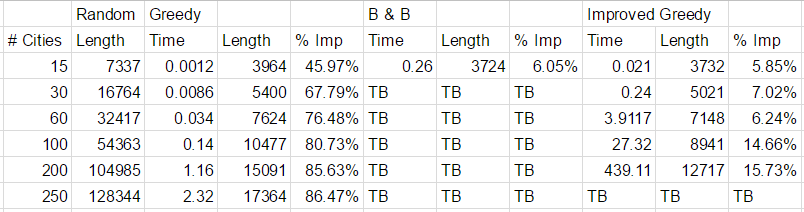
The graph follows a curve approximately similar to O(n2log n), encouraging our theoretical estimate.

**4 Analysis**

**4.1 Methodology**

The table at the bottom of the page shows the runtimes, path costs, and percent improved for each algorithm at various city scenarios. To record this data, each algorithm was run on each size five times and then the average was recorded.

**4.2 Observations**

It is clear that the improved greedy algorithm did indeed improve upon the original greedy algorithm. There were significant time costs, especially as the number of cities grew, but the lengths still improved. Depending on real life applications of these algorithms, that may or may not be worth the costs. For the 200 city scenario, if the 15% improvement is worth 375% loss in calculation time, than the improved greedy algorithm would be beneficial. Otherwise, the 15% improvement may not be valuable enough to justify the significant time loss. It is also interesting to consider the possibility that the branch and bound may have been able to retrieve better paths, if it had been improved to the point where it could calculate large sets in reasonable times. We determined that less than ten minutes was a reasonable runtime for each situation, but the branch and bound could not complete any scenario greater than fifteen cities.

**5 Moving Forward**

There are many ways that this improved greedy algorithm could be better optimized. If the runtimes could be reduced, this algorithm could become much more beneficial. One way to optimize would be to determine quicker ways to reduce the numbe of starting points. The way the algorithm currently runs, it cycles through every city as a starting point, and then calculates the path from there. It would significantly reduce runtimes if some sort of pruning mechanism was created to remove some of the starting points. This could possibly be done by running the original greedy algorithm on each starting point, creating a priority queue of these starting points based on the calculated path lengths, and then running the improved greedy algorithm on a determined subset of the queue. This queue could be trimmed by either size – determining a number or percent of the total size to run on – or by a cost limit – determining a maximum cost and not running on the portion of the queue that has a greater cost than that limit.