Optimal Solution-Generating Algorithms for the Traveling Salesman Problem using Google Maps

Josh Kautz
kautz035@umn.edu
May 7, 2015

CONTENTS

1 - Abstract 1
2 - What is the Traveling Salesman Problem? 2
3 - History of the TSP 2
4 - Application of the TSP 3
5 - Theory of Solving the TSP 4
6 - Using Google Maps to Solve the TSP 5
7 - Results 6
8 - Conclusion 7

1 - ABSTRACT

Google Maps is the largest interactive web cartography service made free for public use, where users may find the optimal route to travel from one location to another. However, this service fails to address the use case where a user wishes to find the optimal route for visiting
multiple locations. This problem is a very well known problem in the realm of computational mathematics and computer science, is commonly known as the Traveling Salesman Problem [21]. There is no efficient way to optimally solve the Traveling Salesman Problem, but there are many ways to arrive at a sub-optimal solution [18]. In this study, I find which implementations of algorithms consistently yield the most accurate tours of the TSP using Google Maps driving routes.

2 - WHAT IS THE TRAVELING SALESMAN PROBLEM?

The Traveling Salesman Problem (TSP) is arguably one of the most extensively studied problems in computational mathematics. The problem, given a collection of cities and the cost of travel between each pair of them, is to find the cheapest way of visiting all of the cities and returning to the starting point [3][16]. In a majority of the versions of the TSP that are studied, the travel costs are said to be symmetric; if there are two cities, traveling from one city to the second costs just as much as traveling from the second city to the first. While this is very common in theoretical instances of the problem, it is far less common in real world applications of trying to optimize routes for travel [5]. The problem can also be explained and visualized using graphs, where the data consists of weights assigned to edges of a finite, complete graph, and the goal is to find a cycle with the minimum total weight. In the context of the TSP, Hamiltonian Cycles (a closed loop passing through all the vertices) are commonly called tours [17].

It is very easy to calculate the number of different possible tours through \( \eta \) number of cities. After the starting city, there are \( \eta - 1 \) choices for the second city, \( \eta - 2 \) choices for the third city, and so on. By multiplying these values together, the result is always equal to \( (\eta - 1)! = (\eta - 1) \times (\eta - 2) \times (\eta - 3) \ldots 3 \times 2 \times 1 \). Since the cost of travel does not depend on the direction of a tour in a symmetric TSP, the total number of tours can be cut in half to yield \( (\eta - 1)! \div 2 \) possible tours. However, this is only applicable to symmetric TSP situations. Regardless, this still quickly produces a very large number of possibilities for tours, which makes the problem increasingly difficult to solve optimally; this has historically been claimed to be the reason why the TSP seems so difficult to solve. The monstrous rate of growth of \( (\eta - 1)! \div 2 \) rules out the possibility of checking all tours one by one as the number of cities increases. This number only increases when trying to apply the TSP to the real world where dealing with modern methods of transportation, such as one-way roads, causes the problem to be asymmetric.

3 - HISTORY OF THE TSP

The Traveling Salesman Problem has been around for many decades and the origins of the problem are uncertain. In the 1920's, an economist and mathematician in Vienna by the name of Karl Menger publicized the TSP among his colleagues. Later, in the 1930's the problem became a topic of discussion in the mathematics department of Princeton [11]. By the 1940's, the TSP was being studied by statisticians in extension to real-world agricultural ap-
applications; since then, the problem has been studied extensively by computer scientists and mathematicians for combinatorial optimization. Examining the tours produced by the TSP one by one is not a realistic approach because of the large size, but for a very long time no other alternatives existed. Over the last few decades, computer codes for the TSP have become impressively sophisticated, and a sign of these improvements is the increasing size of nontrivial instances that are being solved [12]. In 1954, a 49-city solution was produced, and 50 years later, in 2004, a 24,978-city solution was produced. The solution that was created in 1954 showed a tour between 49 American cities, which was an impressive size of computation at the time (Figure 3.1). The revolutionary solution that was produced in 2004 was for an optimal tour of Sweden, where 24,978 cities would be visited (Figure 3.2) [1].

4 - Application of the TSP

A majority of the work being done on the TSP stems from the possibilities of its general methods which can be applied to a much wider range of optimization problems. However, by itself, the TSP still does find application in many fields of work. The TSP is a very common problem in many transportation and logistically-oriented applications. One common example of the TSP in the real world is how a school district arranges school bus routes to pick up students; this particular example was very relevant in the 1930’s while the problem was gaining pop-
ularity and momentum, since automobiles were growing increasingly familiar in the United States [2].

More modern applications of this problem involve the scheduling the delivery of meals to homebound persons, the planning of paper routes for newsboys to deliver to their customers, the routing of vehicles for the United States Postal Service for postal pickup and delivery, the routing of a truck to collect coins from payphones in a city, and many others. Political campaign managers who are planning tours for their candidates are also very interested in optimizing routes to save on both time spent on the road and distance traveled, which correlates with costs devoted to gas consumption. Transportation might be the most natural application for the TSP, but the model has led to many other interesting applications outside of a transit setting. One particular example is the programming of a machine to drill holes in a circuit board in the most logical order [1].

5 - Theory of Solving the TSP

The traveling salesman problem looks for an optimal tour through a specified set of cities. To solve a particular instance of the problem, it is essential to find a shortest tour, and then verify that no better tour exists, which is a very difficult task. Finding a solution to an instance of the TSP does not mean finding a pretty good tour, or finding one that is slightly better than a previously known tour; to solve a TSP, one must find the absolute shortest tour and be able to know that no better tour exists. In reference to the previously cited Sweden tour, the complexity of the calculations make it unlikely that a proof can be constructed to check the tour's optimality without the aid of a computer. The study published with the Sweden TSP computation claimed that 84.8 CPU years were used to verify that the best tour had been created [1]. Even though this problem is very complex from a computational standpoint, there are many heuristics and methods known to help completely solve instances with tens of thousands of cities. With the right heuristics and methods, it is even possible to approximate problems with millions of cities within a small fraction of 2-3% accuracy [18].

Traditional methods of solving NP-hard problems, like the TSP, first call for creating an algorithm to find exact solutions. Then, heuristic algorithms are applied to find seemingly good solutions, but which cannot be proved to be optimal. Obviously the most direct solution would be to use brute force by taking all ordered combinations and try each, seeing which one turns out to be cheapest [13]. However, the run time for this approach would be a polynomial factor of approximately $O(n!)$. This number builds up very quickly and is not realistic, even for instances where there are only 25 cities.

For quickly creating an initial solution, the nearest neighbor algorithm, which in the context of the TSP is a greedy algorithm, yields a surprisingly effective short route for how it works. This algorithm simply chooses the nearest unvisited city as the next move. For $n$ cities in the problem, the algorithm tends to, on average, create a path that is only 25% longer
Another algorithm of relevance is the pairwise exchange, where two edges are removed and replaced with two different edges that create a new, slightly shorter tour; this is also known as a 2-opt technique, and is seen in Figure 5.1 [9].

6 - Using Google Maps to Solve the TSP

For this study, a modern approach was taken to use relevant technologies to attempt to solve the TSP. There are many powerful interactive web cartography services, however, none are capable of suggesting optimal (or even efficient) routes based off a list of locations; there are many uses for a feature of this nature, for instance, if someone had a list of stops to make on their way home from work and they were trying to find the quickest route. I took it upon myself to implement multiple algorithmic searches on existing web technologies to solve the TSP of different sizes, and analyze the data to see what could be learned from the results.

The decision to use Google Maps instead of OpenStreetMaps was not an easy one to make. There were pros and cons for both web services, and while I am a very strong advocate of the open source initiative, there were simply too many reasons to use Google Maps in this particular instance [8]. The main benefit of working with Google Maps was the extensive existing documentation and code that littered the Internet [6][10][20]. I was able to find an existing GitHub repository where a fellow had started doing what I wished to do, but his code was no longer compatible with the current version of Maps API; this was extremely useful in the development of cycle of my program [4]. However, the main problem with Google Maps was the usage limits, where a quota was placed on the free Geocode API, which placed a limit of 2,500 requests per 24 hour period. Initially, this severely hindered the testing phase, where a simple test of $\eta = 7$ would create more than double the daily request quota if the algorithm were to check every single tour by brute force. Later it was discovered that though the Geocode API was nice to work with, it ultimately was not necessarily, and we could create the web application without it. The decision to use the Google Maps JavaScript API, rather than the Google Maps Engine API Client Library for Java, was simply due to my familiarity with JavaScript and web programming. Finally, before committing myself to using Google Maps, I made sure to read the Google Maps API terms thoroughly before I signed up to make sure there was noth-


Deciding on how to measure successfulness is difficult for this study, since determining which algorithm works best depends both on the time required for computation and the accuracy of the solution that is generated. Ultimately, the time required to compute a solution depends on the users computer hardware and Internet connection, and due to the sub-optimal conditions I found myself dealing with for both of these categories, I decided to focus entirely on the accuracy of the tour solution that the algorithms produced. The compiled results of this

<table>
<thead>
<tr>
<th>Locations</th>
<th>Nearest Neighbor</th>
<th>2-Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>η = 11</td>
<td>80%</td>
<td>100%</td>
</tr>
<tr>
<td>η = 10</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>η = 9</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>η = 8</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>η = 7</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 7.1: Results
study are found in Table 7.1. These results indicate that some heuristics are better than others for the TSP, but really it depends on the number of locations $\eta$ that will be used. The values in the table indicate what percent of the time the algorithm will yield the optimal solution; the values in the table were found based on 50 sample test trials of each algorithm using the corresponding number of locations.

I was not able to determine how accurate the algorithms were with location sizes $\eta \geq 12$ because we simply did not have the computational means to calculate the optimal route using a brute force approach to check every tour. Judging by the rate of decrease seen in the greedy algorithm results, it may be safe to say that it would experience an alarmingly steep plummet in it's accuracy. It would also not be surprising to see the 2-Opt technique begin to experience some shakiness in accuracy.

8 - CONCLUSION

It is very apparent that without the proper computational power, the TSP quickly becomes a very overwhelming problem to solve, unless you're willing to settle for a slightly sub-optimal solution. There are many methods and heuristics that can be applied to a solution algorithm to calculate a very close answer, but without previous studies it would be hard to know that for sure.

Recently there has been a lot of attention circulating the TSP with researchers in cognitive psychology fields [15]. Some relatively new studies have come forward that suggest humans are able to produce very good quality solutions for the TSP quickly. Scientists have suggested that computer performance on the TSP could very well be improved by being able to understand the methods used by humans for these problems [14]. This could be a very interesting area for expansion and continuation on this particular study.

I do not know the first thing about how Google's internal systems work, but this study has led me to believe that there is little stopping them from making an optimal routing feature in future iterations of their Maps service. They could at least have it be an available feature for a small number of locations without requiring much dedication from their servers.

For this study, I have placed my code online in a public University of Minnesota GitHub repository so that others may look at and build off of my work. I also placed an MIT License on the work, since some of the code I referenced was under an MIT License of its own [19].
References


