Traveling Salesman Experiment

Team JS: Jonathan Irvin Gunawan, Khor Shi-Jie
November 3, 2014

1 INTRODUCTION

The study of traveling salesman problem (TSP) is a natural generalization of the Hamiltonian cycle problem. We are interested to identify a cycle within a weighted graph such that the cycle includes all vertices in the graph and each vertex is only visited once in the cycle. In addition, we would to identify the Hamiltonian cycle with the minimum sum of edge weight. A close analogue to this problem is the Chinese postman problem which seeks to identify a tour of minimum weight that traverses each edge in the graph at least once. While the exact solution to the Chinese postman problem can be obtained in polynomial time, it is surprising that a subtle change in the requirement of the problem makes the Travelling Salesman Problem unsolvable in polynomial time. In particular, we note that

Theorem 1. *TSP is NP-Hard.*

Proof. We will reduce a directed Hamiltonian Cycle Problem into TSP. Given a directed graph G, we construct a weighted complete directed graph G' which has the same vertices as G. In addition, we assign a weight of 1 to the edges in G which were originally in G', and a weight of ∞ otherwise. Suppose we found a solution to the TSP using graph G' that has finite weight. Then, each of the edges must have weight 1 and hence must be present in the original graph G. As such, the solution to the TSP in G' is also a solution to the Hamiltonian Cycle Problem in G. On the other hand, if the solution to TSP in G' has infinite weight, then there is no Hamiltonian cycle which only consist of edge weight 1. Hence, there is no Hamiltonian cycle in G. As such, the directed Hamiltonian Cycle Problem reduces to TSP. Since the directed Hamiltonian Cycle problem is NP-Hard, TSP is also NP-Hard. □

There has been considerable research done in the past in determining the exact solution for TSP in a graph. A naive algorithm that checks all possible Hamiltonian cycle will incur $O(n!)$ time. It is possible to improve the runtime to $O(n^2 2^n)$ using dynamic programming (i.e. Held-Karp algorithm), but even such algorithm will take significant time on graphs with small number of vertices[1]. Researchers have explored other approaches such as branch-and-bound algorithms and linear programming. Currently, the record for the highest number of vertices in a graph in which a TSP solution is found

[1] On a modern computer, it is possible to obtain the optimal solution using Held-Karp algorithm within 1 second for a graph with less than 18 nodes. [4] Any more vertices will incur a much higher runtime.

is approximately 85,900 nodes[2]. Nevertheless, this is a small quantity compared to many real world instances of TSP.

Due to the impossibility of find a polynomial-time algorithm to solve TSP and the relevance of TSP to real life problems, there has been a lot of research that seeks to find an efficient approximation algorithm with a small approximation ratio. Within our course, we have explored several approximation algorithm to solve the TSP, including a 2-approximation algorithm[3] based on constructing a minimum spanning tree (MST), and a 1.5-approximation algorithm that include the application of a minimum weight perfect matching algorithm. However, the effectiveness of these algorithms are only theoretical in nature. While the worst-case bounds for these algorithms are irrefutable, the optimality of these algorithm may differ from our expectations when we apply them to real world data (i.e. a 2-approximation algorithm may perform better on average than a 1.5-approximation algorithm). As such, this project aims to compare the suitability of various approximation algorithm when applied to graphs obtained from real world data or other random means.

2 Problem Formulation

Generally, there are four variants of TSP that one can seek to solve. We can either have a graph in which the distance function is a metric, or a graph with a generic distance function. In addition, we can also choose to allow repeated vertices within our solution. The four variants are summarized in the table below:

	Repeats	No Repeats
Metric	M-R TSP	M-NR TSP
General	G-R TSP	G-NR TSP

We have shown in class that M-R TSP, M-NR TSP and G-R TSP are equivalent. For our project, we are more interested in M-NR TSP as we are using data obtained from the real world. In addition, we will assume that the graph is non-directional, and the distance between any two vertices is well-defined.

For this project, we will implement four different approximation algorithms and profile the performances on a set of random graphs and real world data. The four approximation algorithms are:

1. *Nearest neighbour heuristics.* We start from a cycle with only one vertex in the cycle. Then, while there are vertices which are not included within our cycle, we choose a vertex which is closest our cycle. Then, we add the vertex to the cycle at a position whereby the increase in the length of the cycle is minimal. Once all the vertices are added to the cycle, we will obtain a $O(logn)$-approximate solution to the TSP (to be proven in the next section).

2. *Minimum spanning tree.* We will build a minimum spanning tree for the graph and conduct a DFS traversal on the graph. Within the DFS traversal, we will skip any vertices which has already been visited. This will produce a 2-approximate solution to the TSP (proven in class).

3. *1.5-approximate algorithm.* After finding the minimum spanning tree of the graph, instead of conducting a DFS, we add edges to the graph until all vertices in the graph have even degree. Then, we can find an Eulerian tour for the graph. The edges to be added will be selecting using a minimum weight perfect matching algorithm such as a modified Edmond's Blossom algorithm.[5]

[2] Found by Applegate et al. using Concorde TSP Solver. [1]

[3] This algorithm only applies for a metric TSP. The same algorithm can be applied to obtain approximate solutions to M-R TSP and G-R TSP as these variants are equivalent.

4. *2-opt algorithm.* This algorithm makes use of the fact that a cycle without crossing is shorter than a cycle with crossing under the assumption of triangle inequality. We will refine any random cycle until there can be no more improvement to the weights. This algorithm is an $O(\sqrt{n})$-approximate algorithm on average case. [3]

We are interested in the optimality of the solution produced by the above four algorithms, as well as the runtime in producing the solutions. In particular, we want to know how well these algorithms will perform on random graphs and real world data.

In our project, we will first implement methods to generate random graphs and prepare 5 such graphs for experimentation. Then, we will parse real world data from (to be confirmed) and (to be confirmed) and store them in a weighted graph. We will apply each approximation algorithm on our generated graphs and obtain the weight of TSP solution produced by each algorithm. Each algorithm will be executed thrice and the average runtime taken for the algorithms will be calculated. Upon collecting these data, we will compare the results and conclude on the effectiveness of each approximation algorithm.

3 Algorithms and Theory

In this section, we will describe the implementation of the four approximation algorithms proposed in the previous section. We will assume that the graph is stored as an adjacency list.

3.1 Nearest Neighbour Heuristic

Algorithm 1 Nearest Neighbour Heursitics

```
1:  procedure NEAREST-NEIGHBOUR-HEURISTICS(g)        ▷ Returns the TSP approximation
2:      Set visited[v] ← false, for all v ∈ V
3:      TSP ← ∅
4:      u ← 0
5:      visited[u] ← true
6:      while there is unvisited vertex do
7:          v ← unvisited vertex that is closest to u
8:          TSP ← TSP ∪ (u, v)
9:          visited[v] ← true
10:         u ← v
11:     return TSP
```

Theorem 2. *Algorithm 1 is $O(logn)$ approximation*

Proof. Shi-Jie is handsome <3 □

References

[1] D. L. Applegate, R. M. Bixby, V. Chavatal, and W. J. Cook. *The Traveling Salesman Problem*. Princeton University Press, 2007.

[2] J. Clark and D. Holton. *A First Look at Graph Theory*. World Scientific, 2005.

[3] C. Engels and B. Manthey. Average-Case Approximation Ratio of the 2-Opt Algorithm for the TSP. *Operations Research Letters*, 2009. Retrieved from `http://wwwhome.math.utwente.nl/~mantheyb/journals/ORL_EngelsManthey_2Opt.pdf`.

[4] S. Halim and F. Halim. *Competitve Programming 3*. lulu, 2013.

[5] V. Kolmogorov. Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 2009. Retrieved from `http://pub.ist.ac.at/~vnk/papers/blossom5.pdf`.