Bangladesh University of
Engineering and Technology

Technical Writing and Presentation

CSE 300

The Traveling Salesman Problem

Prepared By:

Zahin Wahab, 1505031.
Mursalin Habib, 1505049.

July 10, 2020

Abstract

In this report we take a look at a famous computational problem called the *Traveling Salesman Problem (TSP)*. For roughly 70 years, the TSP has served as the best kind of challenge problem, motivating many different general approaches to coping with NP-hard optimization problems.

More specifically, we look at the general problem statement and some approaches to solving it. We also talk about its intractability and how it is even hard to approximate. In the final sections, we focus on some special cases of the TSP (such as metric TSP) and provide some good approximation algorithms for them.

1 Introduction

Imagine that you are a salesperson who needs to visit a set of cities to sell their goods. You know how many cities you need to go to and the distances between each city. In what order should you visit each city exactly once so that you minimize your travel time and so that you end your journey in your city of origin?

This problem is what is known as the *Traveling Salesman Problem*. It is a problem that has been the bane of existence of computer scientists for more than 70 years. Although the problem itself is very easy to state, as we will see, it is notoriously difficult to find an efficient solution to it.

2 Problem Statement

More formally speaking, the input to the *Traveling Salesman Problem* is a complete undirected graph $G = (V, E)$, with a non-negative cost $c_e \geq 0$ for each edge $e \in E$. By a TSP *tour*, we mean a simple cycle that visits each vertex exactly once. The goal is to compute the TSP tour with the minimum total cost. The cost of a tour is simply the sum of the costs of all the edges that make up the tour. For example, in Figure 1, the optimal objective function value is 13.

Although, the TSP gets its name from a silly story about a salesperson who has to make a number of stops, and wants to visit them all in an optimal order, it definitely comes up in a lot of real-world scenarios. In fact, a lot of

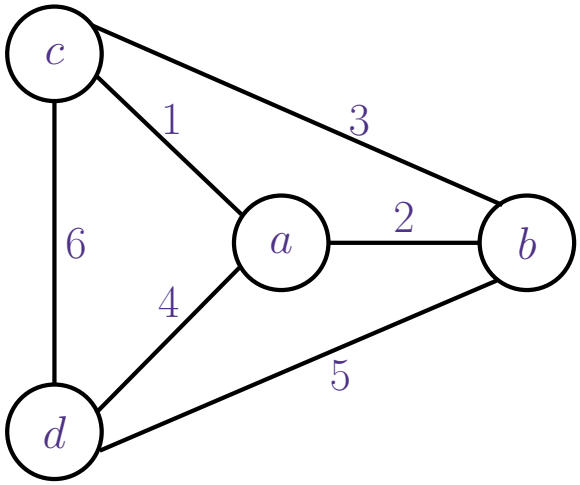

Figure 1: Example TSP graph. Best TSP tour is *a*-*c*-*b*-*d*-*a* with cost 13.

real word problems are just thinly disguised versions of the TSP. For example, suppose a number of tasks need to get done, and between two tasks there is a setup cost (from, say, setting up different equipment or locating different workers). Choosing the order of operations so that the tasks get done as soon as possible is exactly the TSP. Or think about a scenario where a disk has a number of outstanding read requests; figuring out the optimal order in which to serve them again corresponds to the TSP.

3 Approaches to Solving the Problem

3.1 The Brute Force Approach

When you are faced with a problem, the very first thing you are inclined to do is to try a brute force algorithm. A brute force algorithm for the TSP would be extremely simple. It would go something like this:

- Find all possible tours of the graph.
- Compute their costs.
- Pick the tour with the minimum cost.

So, how good is our very easy-to-state brute force algorithm? There are a total of $(n-1)!$ tours of a complete graph on n vertices. This is because each tour is essentially a cyclic permutation of the vertices. Computing the cost of such a tour takes time linear in the number of vertices. So, our brute force algorithm has a running time of $O(n!)$.

3.2 Other Approaches to Solving the General TSP

The brute force approach does a terrible job at solving the general TSP. There are other approaches that do a bit better, but not much. For example, the Held-Karp algorithm takes a dynamic programming approach to solving the TSP. It has a running time of $O(n^2 2^n)$. This is an improvement over our brute force approach, but it doesn't really scream 'efficient'.

What we want is a *polynomial* time algorithm for the general TSP. However, the existence of such an algorithm does not seem likely as the next section will show.

4 Intractability of the TSP

As stated before, for roughly 70 years, the TSP has served as the best kind of challenge problem. People like George Dantzig have spent a fair amount of time trying to solve ever-bigger instances of TSP. Due to numerous failed attempts at finding a polynomial time algorithm, Jack Edmonds, in a remarkable paper titled *Paths, Trees and Flowers*, claimed that there can not exist a polynomial time algorithm for the Travelling Salesman Problem. His conjecture, which he made in 1965, remains unresolved to this very day.

When Edmonds made his claim, there was no notion of an NP-complete problem. Now we know that the TSP is NP-complete. So, we do not expect to find an exact algorithm for it that also runs in polynomial time. However, even before the development of NP-completeness in 1971, experts were aware that the TSP is a "hard" problem in some sense of the word.

5 Coping with NP-Completeness

Now we know that the Traveling Salesman Problem is NP-complete. But what can we do about it? The bad news is, *knowing* that a problem is NP-

complete does not make it go away. In the real world, when you are faced with such a problem, you can not just run away from it.

There are a number of general techniques of dealing with NP-completeness. We talk about two such techniques here. They are:

- **Approximation Algorithms:** these are algorithms that run in polynomial time but does not always produce the optimal result. In other words, we relax correctness to get fast results.
- **Solving Special Cases:** Sometimes we do not care about all possible instances. And the instances we do care about happen to be tractable.

Let's take a look at both of these.

5.1 Approximation Algorithms for the General TSP

Unfortunately, the Traveling Salesman Problem is hard to even approximate. More specifically,

Theorem 1. *Unless $P = NP$, there is no α-approximation algorithm for the TSP (for any α).*

Recall that an α-approximation algorithm for a minimization problem runs in polynomial time and always returns a feasible solution with cost at most α times the minimum possible.

We present a proof of this theorem. The main idea is to do a reduction. What we wish to show is that if we had an α-approximation algorithm for TSP, we could use it solve some other NP-complete problem, say the Hamiltonian Cycle problem. Here are the details.

Proof. We prove the theorem using a reduction from the Hamiltonian cycle problem. The Hamiltonian cycle problem is: given an undirected graph, does it contain a simple cycle that visits every vertex exactly once?

For the reduction, we need to show how to use a good TSP approximation algorithm to solve the Hamiltonian cycle problem. Given an instance $G = (V, E)$ of the latter problem, we transform it into an instance $G' = (V', E', c)$ of TSP, where:

- $V' = V$

- E' is all edges (so (V', E') is the complete graph);
- for each $e \in E'$, set $c_e = \begin{cases} 1 & \text{if } e \in E \\ > \alpha n & \text{if } e \notin E \end{cases}$

The key point is that there is a one-to-one correspondence between the Hamiltonian cycles of G and the TSP tours of G' that use only unit-cost edges. Thus:

(i) If G' has a Hamiltonian cycle, then there is a TSP tour with total cost n.

(ii) If G' has no Hamiltonian cycle, then every TSP tour has cost larger than αn .

Now suppose there were an α-approximation algorithm $\mathcal{A}$ for the TSP. We could use $\mathcal{A}$ to solve the Hamiltonian cycle problem: given an instance G of the problem, run the reduction above and then invoke $\mathcal{A}$ on the produced TSP instance. Since there is more than an α factor gap between cases (i) and (ii) and $\mathcal{A}$ is an α-approximation algorithm, the output of $\mathcal{A}$ indicates whether or not G is Hamiltonian. (If yes, then it must return a TSP tour with cost at most αn; if no, then it can only return a TSP tour with cost bigger than αn.) □

5.2 Solving Special Cases

From the section above, we see that even approximate solutions to the TSP are infeasible if we focus on solving the general case. This forces us to look at special cases.

What are some special cases that we might be interested in? One such case is *metric TSP*.

In metric TSP, we assume that the edge costs satisfy the triangle inequality (with $c_{uw} \leq c_{uv} + c_{vw}$ for all $u, v, w \in V$). In other words, metric TSP deals with graphs, where the shortest path between two vertices is the one-hop path between them. In applications, where edge costs model real world distances, this is the special case we should be focusing on.

Unfortunately, even in this special case, the problem is NP-complete. The proof for this is extremely similar to the proof of the theorem in the preceding section. So, we don't repeat ourselves for the sake of brevity.

There is, however, light at the end of the tunnel! Instead of trying to solve metric TSP exactly, we can try solving it *approximately*. This is where humanity's algorithmic perseverance shines through. We have a bunch of very good approximation algorithms for metric TSP. We describe two of them here.

5.2.1 The MST-heuristic Algorithm

The main idea behind the MST-heuristic algorithm is this: no matter what the cost of the optimal TSP tour is, it can't be smaller than the cost of the minimum spanning tree of the same graph. So, if we can produce a tour that has cost close to that of the MST, we can not be too worse off.

Here is an outline of the MST-heuristic.

MST Heuristic for Metric TSP

construct the graph H by doubling every edge of T
compute an Euler tour C of H
`// every v ∈ V is visited at least once in C`
shortcut repeated occurrences of vertices in C to obtain a TSP tour

The MST heuristic is actually a 2-approximation algorithm for metric TSP. This is because the tour produced by it has cost at most twice of that of the MST of the graph.

5.2.2 Christofides's Algorithm

Christofides's algorithm works because of the same principle as the MST heuristic. But instead of doubling each edge of the MST to find an Eulerian tour, we look for a minimum cost perfect matching.

We will not go into all the details. But we do state the final result.

Theorem 2. *Christofides's Algorithm has an approximation ratio of* $\frac{3}{2}$.

Christofides's algorithm is from 1976. Amazingly, to this day we still don't know whether or not there is an approximation algorithm for metric TSP better than Christofides's algorithm. It's possible that no such algorithm exists (assuming $P \neq NP$, since if $P = NP$ the problem can be solved optimally in polynomial time), but it is widely conjecture that $\frac{4}{3}$ (if not better) is possible. This is one of the biggest open questions in the field of approximation algorithms.

6 Conclusion

We have taken a look at the Traveling Salesman Problem, its history and tried multiple approaches to solving it efficiently. We have noted that despite being a very well-studied problem, there is a lot of open questions revolving it. Specially, in the field of approximation algorithms, not much progress has been made in the span of the last 40 years. While we understand that this fact may come off as depressing, we feel the prospect in and of itself is very fascinating. It means that there is still a lot of things left to explore. And we find that thought very exciting.