The NP-hard problem: *Traveling Salesman Problem*

Cheng Huang

University of Chinese Academy of Sciences
huangcheng18s@ict.ac.cn

July 26, 2019

Abstract

This document presents some basic information about traveling salesman problem (TSP). TSP is a classic NP-hard problem in graph theory, with many applications such as traffic scheduling, DNA sequencing, etc. Due to its NP-hardness, exact algorithms can only solve small-size problems (less than 10k cities). Meanwhile, there are a lot of heuristic algorithms like Lin-Kernighan heuristics, and TSP with millions of cities can be solved approximately now (within 2-3% of an optimal tour).

I. Definition and Applications

Imagine you are a selling agent of Loongson PCs and laptops. You want to travel around China to convince the leaders of the local governments to buy your products. You have to meet them in person to fully introduce the advantages of your products (e.g. security is far more important than performance for government usage) and to show your sincerity. There are 34 provicial-level divisions in the whole country (Figure 1). There is a certain cost to travel between each pair of them, roughly related to the distance. So, how will you arrange your whole travel to minimize the cost and eventually go back to your start point (e.g., Beijing). This problem is essentially a traveling salesman problem (TSP).

Here is a formal definition of TSP: Given an undirected graph $G = (V, E)$, $V = \{v_i \mid i = 1, \cdots, n\}$ is the set of vertices, $E = \{(v_i, v_j, w_{ij}) \mid i, j = 1, \cdots, n \text{ and } i < j\}$ is the set of edges, where w_{ij} is the weight (cost) of edge (v_i, v_j). Find a Hamiltonian cycle of G with a minimum cost, i.e., find a cycle of G that covers all vertices in V with a minimum cost. TSP is an NP-hard problem. We will talk about this in detail in section II.

To simplify the problem, we assume graphs

Figure 1: *Provincial-level divisions of China. [Wikipedia contributors, 2019]*

in this article are **complete**, which means for each pair of vertices, an edge exists. There are always some methods to covert other graphs to a complete one with identical TSP solutions.

There are several common variations of TSP, like metric TSP. In metric TSP, costs of edges satisfy triangle inequality. That is, for 3 cities A, B and C, the direct connection from A to B is never farther than a route via C:

$$d_{AB} \leq d_{AC} + d_{CB}$$

There are many alternatives for distance, or costs, such as Euclidean distance, or Manhattan

distance (sum of the absolute value of difference in each coordinate) etc. Metric TSP is also NP-hard except for Euclidean TSP with rational coordinates and discretized distance, which is NP-complete.[Papadimitriou, 1977] We will not go farther about this. Another example of variation is asymmetric TSP, corresponding to a directed graph. Asymmetric TSP seems more practical. It can be transformed to a equivalent original (symmetric) TSP by adding a ghost node to each real node, with a negative cost edge between them.[Kumar, 1996] And there is path-TSP (corresponding to original tour-TSP) that requires a Hamilton path instead of cycle. Path-TSP and tour-TSP reduce to each other in linear time.[Papadimitriou, 1977] And finally there is multiple TSP with more than one salesman, which is apparently useful in vehicle routing and other problems.[Bektas, 2006]

TSP has many practical applications as follows. Some of them are also related to its variations.

- TSP as a mathematical problem first comes from a school bus routing problem.[Zambito, 2006] As its name implies, many problems involving traveling can be considered as a TSP in practice. For example, a businessman may have several trips to different destinations on coming. How to make a travel plan with a minimum cost? Similar examples such as a band making a performing tour plan, a postman/deliverer deciding the order of delivery, or projectionists from Culture Department playing movies in rural areas in the good old days.
- In the examples above, vertices and edges in a graph correspond to cities (or villages) and distances between each other, respectively. It can also be applied to outer space satellites. To observe several different stars and take high definition pictures, satellite with a telescope need to adjust its gesture and location. How to manage the adjustment to minimize fuel consumption? This is also a problem can be solved by applying TSP.[Guo, 2006]
- In a production line, there are machines drilling holes on circuit boards or something else. If the machine needs to drill dozens or more holes on one board, it's movement needs to be arranged using TSP to save energy. Then a large amount of money can be saved, after months or years.[Zambito, 2006]
- TSP is also applied in genome researches. According to [Agarwala, 2000], they use TSP to construct radiation hybrid (RH) maps from data on different RH panels, which performs better and faster than specially tailored RH mapping software. What is interesting is on the contrary, genetic algorithm is one of the widely used heuristic techniques to solve TSP, for which we will come back at section IV.
- There are other applications such as wire arrangement, crystal structure analysis, clustering and so on.

In the following sections, we will first discuss TSP's computational complexity, and proof its NP-hardness by reduction. Then we'll introduce different algorithms, including exact algorithms and approximation ones, to solve TSP. After all this, we will make a brief summary to put an end to this article.

II. Proof of NP-hardness

We will first introduce some basic concepts of computational complexity theory:

P (or PTIME) stands for the class of problems which can be solved using polynomial time on a deterministic Turing machine (or a PC). Roughly speaking, a problem in P can always be solved by an efficient algorithm, or is solvable. The time complexity to solve a P problem is $O(n^{O(1)})$.

NP stands for the class of problems which can be solved using polynomial time on a non-deterministic Turing machine. Non-deterministic Turing machine has the ability of executing all the possible execution paths in parallel. You can image an execution tree of a problem. Leaf nodes of this tree has a state of accept or reject. A non-deterministic Turing machine can then explore this tree layer by

layer (all nodes in the same layer are visited in parallel), while a deterministic Turing machine can only explore it node by node. Putting it another way, an NP problem can be verified in polynomial time. That is, given one possible solution of the problem, we can decide whether it is a right solution or not in polynomial time on a deterministic Turing machine. The verification is actually running through one path of the execution tree, to see whether it halts at an accept state or not. P is a subset of NP.

It is very clear that problems in P and NP are mainly **decision** problems. A decision problem asks whether a solution exists, while a **optimization** problem asks to maximize/minimize some objective.

An NP-complete problem is first in NP. Besides, all other problems in NP can be reduced to an NP-complete problem in polynomial time. The first proofed NP-complete problem is SAT.[Cook, 1971] After that, to proof an NP problem is NP-complete, researchers only need to reduce some known NP-complete problem to it.

At last, problems in NP-hard class need to be reducible from all the problems in NP using polynomial time. That is, an NP-complete problem can reduce to an NP-hard problem. NP-hardness doesn't require the problem to be in NP, so NP-complete is a subset of NP-hard.

As for TSP, there are two versions of this problem: decision version and optimization version. We only talked about optimization version so far. For decision version, a constant k is given and the question is to decide whether a Hamiltonian cycle with a cost less than k exists.

Lemma 1. *The decision version of TSP is in NP.*

Proof. Given a graph $G = (V, E)$ with $|V| = n$, a constant k, and a possible solution S of TSP (i.e., a sequence of vertices that indicates a path in G). We can check whether S is a Hamilton cycle by transverse it ($O(n)$), and meanwhile adding up all the costs. If S satisfies both conditions, it is a real solution; otherwise not. So we can verify the decision version of TSP in polynomial time, which makes it an NP problem. □

Lemma 2. *NP-complete problem Hamilton Cycle can reduce to the decision version of TSP.*

Proof. This proof is rather trivial. Given a graph G and let k be some number larger than the sum of all edge costs. Suppose there is a Turing Machine M, M takes G and k as input and outputs whether a Hamilton cycle with a cost less than k exists. Because k is larger than all possible paths' costs in G, run M on G and k and the output is identical to the answer of HC. Thus HC can reduce to the decision version of TSP. □

Based on lemmas 1 and 2, the decision version of TSP is in NP-complete.

Theorem 1. *The decision version of TSP is in NP-complete.*

As for the optimization version of TSP, we have lemma 3:

Lemma 3. *The decision version of TSP can reduce to the optimization version of TSP.*

Proof. Given a graph G and a constant k. Suppose there is a Turing Machine M that takes G as input and outputs the Hamilton cycle with a minimum cost. Run M on G and add up edge costs in output cycle. If the sum is less than k, Hamilton cycle satisfies the decision version TSP exist; otherwise not. Thus the decision version of TSP can reduce to the optimization version TSP. □

However the optimization version TSP is not in NP. Given a solution of TSP, it is hard to determine whether it is optimal. As a conclusion, the optimization version of TSP is in NP-hard.

Theorem 2. *The optimization version of TSP is in NP-hard.*

For the rest of this article, when referring to "TSP", we actually mean "the optimization version of TSP".

The NP-hardness of TSP indicates that the known exact algorithms are at least superlinear, so it is impossible to totally solve a problem

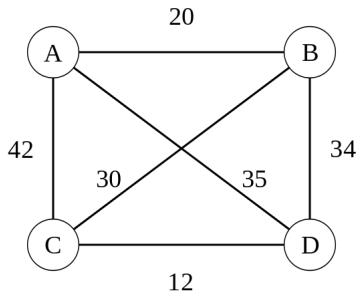

Figure 2: *A 4-node complete undirected graph. [Wikipedia contributors, 2019]*

with a large input. We will present several exact algorithms to solve small-input TSPs and other algorithms to approximately solve large-input TSPs in next sections.

III. Exact Algorithms

In this section and the next, we will use the graph in Figure 2 as a simple example to demonstrate how these algorithm works.

i. Brute Force

Brute force always works. We can try all permutations of vertices and choose the one with minimum cost. In our example, we have

$$A \to B \to C \to D \to A, cost = 97$$
$$A \to B \to D \to C \to A, cost = 108$$
$$A \to C \to B \to D \to A, cost = 141.$$

So the Hamilton cycle with a minimum cost is $A \to B \to C \to D \to A$.

Generally speaking, for an n-node graph, there are $(n-1)!/2$ such permutations. It is worth noting that circular permutations (e.g. $A \to B \to C \to D$ and $D \to A \to B \to C$) and counterclockwise ones (e.g $A \to B \to C \to D$ and $D \to C \to B \to A$) are considered identical, so the total number is $(n-1)!/2$ instead of $n!$. Thus the time complexity of brute force is $O(n!)$, even greater than $O(2^n)$.

Table 1: *The intermediate results of Held-Karp algorithm (with trace back).*

S	$C(S,B)$	$C(S,C)$	$C(S,D)$
$\{B\}$	20(2)	\	\
$\{C\}$	\	42	\
$\{D\}$	\	\	35(1)
$\{B,C\}$	72	50(2)	\
$\{B,D\}$	69	\	54
$\{C,D\}$	\	47(1)	54
$\{B,C,D\}$	77(1)	66	62(2)
back to A	97(1)	108	97(2)

ii. Held-Karp Algorithm

Held-Karp algorithm [Held, 1962] is based on dynamic programming. Given $S \subseteq \{B,C,D\}$ and $X \in S$, let $C(S,X)$ denote the minimum cost of starting from A and visiting all cities in S and terminating at X. Then

$$C(S,X) = \min_{Y \in S \setminus \{X\}} \{C(S \setminus \{X\}, Y) + d_{YX}\}$$
$$C(\{X\},X) = d_{AX}, \text{ for any } X \in S.$$

We need to find an X with minimum $C(\{B,C,D\},X) + d_{AX}$. Then trace back to get the Hamilton cycle we want. In our example, we can construct table 1.

As the table shows, two optimal results come from the clockwise and counterclockwise paths of an identical cycle. We can trace back and get the whole cycle.

For an n-node graph, the table size is $n2^n$. n stands for the number of nodes in horizontal dimension, and 2^n stands for the number of the power set of V (the vertical dimension correspondingly). To fill up one entry of the table, we need to check $n-2$ former entries in the worst case (i.e. when adding the last node to S, we need to go through all $n-2$ nodes that are already in S and choose the optimal one after adding). So the time complexity of Held-Kalp algorithm is $(O(n^2 2^n))$.

One main drawback of dynamic programming is space complexity. The space complexity of Held-Kalp algorithm is $O(n2^n)$ (i.e. table size). As a result, this algorithm cannot be used to solve TSP with a large number of "cities".

iii. Branch-and-cut Algorithm

This algorithm is presented by [Padberg, 1991]. According to Wikipedia [Wikipedia contributors, 2019], this approach holds the current record of solving an instance with 85900 cities.

I tried to read the original paper, but it turns out it is too difficult for me (and 41-page long!). So I'll just stop here as an end of this section.

We will not introduce more exact algorithms here. Improving the time bound of exact algorithms is very difficult. It is undetermined whether an algorithm runs in time $O(1.9999^n)$ exists.[Woeginger, 2003] And that's where approximation algorithms come on the stage.

IV. Approximation Algorithms

i. Constructive Heuristic Algorithms

Nearest Neighbor (NN) is one of the so-called constructive heuristic algorithms. It is quite intuitive: always choose the next node which is the closest. In our example, starting from A, d_{AB} has the smallest cost among d_{AC}, d_{AD} and itself, so AB is added. Similarly, we have BC and CD added. And we add AD at last to construct a cycle. So the total cost of this cycle is 97, which happens to be optimal.

The time complexity of NN is $O(n^2)$: for one node, we have to visit each of other $n-1$ nodes to get its nearest neighbor. According to johnson1997, NN yields a cycle 25% longer than optimal on average. However, in some cases, it is possible to construct a graph that makes NN get the worst route.

There are also some other constructive heuristic algorithms, such as greedy, Clarke-Wright and Christofides. In greedy algorithm, we always add an edge which is available (meaning neither of its end nodes has a degree of 2 or more in this construction) and with a minimum cost in the whole graph, instead of from some certain node's neighbors in NN. In Clarke-Wright algorithm, we arbitrarily choose a node as "hub" and suppose that the "salesman" has to come back to hub after visiting every node. For every pair of non-hub nodes, we can calculate the cost saving by bypassing hub node. We then run greedy algorithm on those cost savings to construct the demand cycle. And Christofides algorithm uses minimum spanning tree and minimum-weight perfect matching to construct a TSP tour. Under triangle inequality (so called metric TSP in section I), Christofides algorithm gives a answer with a cost 2 times the optimal in the worst case.[Johnson, 1997]

ii. Lin-Kernighan Algorithm

Lin-Kernighan algorithm [Lin, 1973] is also called k-opt algorithm. The idea is rather simple: given a tour (randomly or greedily) in a TSP, we first delete k mutually disjoint edges. Then we reassemble the remaining fragments into a new tour. Let's take 2-opt as an example. Suppose we have a cycle $A \to B \to C \to D \to A$ given by NN, we can delete $A \to B$ and $C \to D$ and reassemble to $A \to C \to B \to D \to A$, or delete the other two edges and reassemble to $A \to D \to B \to C \to A$. Neither of these new cycles has a less cost than the original one, so we keep it as the answer.

A similar algorithm is variable-opt (v-opt) algorithm. It is a generalization of the k-opt method. Deleting and reassembling more edges every now and then helps to move the tour out of local minimum.

Lin-Kernighan algorithm family is considered most powerful heuristics to solve TSP. They are even used to solve instances with millions of cities, and the solutions are guaranteed to be within 2-3% of an optimal tour.[Rego, 2011]

iii. Ant Colony System

This is an interesting method models how ants find short paths between food sources and their nest. ACS starts a lot of walks ("ants") on the graph. Each virtual ant chooses next node to visit in some heuristic way, based on the distance and information ("pheromone") left by other virtual ants until it completes a tour.

And eventually, the shorter the tour, the more its pheromone deposits.

iv. Genetic Algorithm

The basic idea of GA is similar to v-opt Lin-Kernighan: find local optimal solution and try to jump out. In GA, we select some nodes as "parents", and then apply some change to them (so-called "mutation" and "crossover". After that, we apply a selection strategy and update the solution if it survives. There are several different ways to fill in this schema. See [Johnson, 1997].

v. Neural Network

Unlike the ascending deep learning technologies these years, neural network is actually a traditional topic in CS and has been studied for decades. Essentially, neural network based TSP algorithms are general integer program solvers. TSP can be formulate to such integer programs. We will not give the full formulation here. The variables x_{ij} in this formulation indicates whether the edge from city i to j is added in solution. Neural network algorithms view x_{ij} as neurons. They are connected in a network that tried to satisfy constraints and meanwhile lower the cost ("energy") in some mathematical way.[Johnson, 1997]

TSP is one of the hottest problem in operational research and combination optimization for last decades. Thus there are lots of other algorithms have been proposed and we cannot go through all of them in this article. So we will stop here and summary the whole article briefly.

V. Summary

In this article, we introduced the basic information about traveling salesman problem. TSP comes from real-world scheduling problems and has lots of other applications in practical. We briefly proofed that the decision version of TSP is in NP-complete class and the optimization version is NP-hard.

There are several exact algorithms to solve TSP, including good old brute force, dynamic programming based Held-Karp, marvelous branch-and-cut algorithm and so on. However, due to TSP's NP-hardness, those algorithms can only solve small instances. Fortunately, there are plenty of brilliant heuristic algorithms that can solve it approximately. Those algorithms include constructive heuristics, Lin-Kernighan algorithm family, ant colony system, genetic algorithm, neural network and so on. Lin-Kernighan based methods is the most powerful ones among them.

References

[Wikipedia contributors, 2019] Wikipedia contributors. (2019, July 18). China. In *Wikipedia, The Free Encyclopedia*. Retrieved 01:55, July 19, 2019, from `https://en.wikipedia.org/w/index.php?title=China&oldid=906855606`.

[Zambito, 2006] Zambito, L. (2006). The traveling salesman problem: a comprehensive survey. *Project for CSE, 4080*.

[Guo, 2006] Guo, J. (2006) An Introduction to Traveling Salesman Problem. *DA ZHONG KE JI,* (8), 229-230.

[Agarwala, 2000] Agarwala, R., Applegate, D. L., Maglott, D., Schuler, G. D., & Schäffer, A. A. (2000). A fast and scalable radiation hybrid map construction and integration strategy. *Genome Research, 10*(3), 350-364.

[Papadimitriou, 1977] Papadimitriou, C. H. (1977). The Euclidean travelling salesman problem is NP-complete. *Theoretical computer science, 4*(3), 237-244.

[Kumar, 1996] Kumar, R., & Li, H. (1996). On asymmetric TSP: Transformation to symmetric TSP and performance bound. *Journal of Operations Research, 6*.

[Bektas, 2006] Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega, 34*(3), 209-219.

[Cook, 1971] Cook, S. A. (1971, May). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (pp. 151-158). ACM.

[Wikipedia contributors, 2019] Wikipedia contributors. (2019, July 14). Travelling salesman problem. In *Wikipedia, The Free Encyclopedia*. Retrieved 03:46, July 25, 2019, from `https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=906214466`.

[Held, 1962] Held, M., & Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics, 10*(1), 196-210.

[Padberg, 1991] Padberg, M.W., & Rinaldi, G. (1991). A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review, 33*, 60-100.

[Woeginger, 2003] Woeginger, G. J. (2003). Exact algorithms for NP-hard problems: A survey. In *Combinatorial optimization—eureka, you shrink!* (pp. 185-207). Springer, Berlin, Heidelberg.

[Johnson, 1997] Johnson, D. S., & McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization, 1*(1), 215-310.

[Rego, 2011] Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research, 211*(3), 427-441.

[Lin, 1973] Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research, 21*(2), 498-516.