Scientific Research Publishing

Applied Mathematics, 2018, 9, 1351-1359
http://www.scirp.org/journal/am
ISSN Online: 2152-7393
ISSN Print: 2152-7385

Fast Algorithm for the Travelling Salesman Problem and the Proof of P = NP

Jinliang Wang

Research Institute for ESMD Method and Its Applications, College of Science, Qingdao University of Technology, Qingdao, China
Email: wangjinliang0811@126.com

How to cite this paper: Wang, J.L. (2018) Fast Algorithm for the Travelling Salesman Problem and the Proof of P = NP. *Applied Mathematics*, **9**, 1351-1359.
https://doi.org/10.4236/am.2018.912088

Received: November 26, 2018
Accepted: December 17, 2018
Published: December 20, 2018

Abstract

In the theory of computational complexity, the travelling salesman problem is a typical one in the **NP** class. With the aid of a brand-new approach named "maximum-deleting method", a fast algorithm is constructed for it with a polynomial time of biquadrate, which greatly reduces the computational complexity. Since this problem is also **NP**-complete, as a corollary, **P = NP** is proved to be true. It indicates the crack of the well-known open problem named "**P** versus **NP**".

Keywords

Travelling Salesman Problem, P versus NP Problem, NP-Complete, Computational Complexity, Maximum-Deleting Method

1. Introduction

The travelling salesman problem asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" [1]. In the theory of computational complexity, this problem is a typical one in the **NP** class [1] [2]. The closely related suspense is the well-known "**P** versus **NP** problem", that is, to determine whether every language accepted by some nondeterministic algorithm in polynomial time is also accepted by some deterministic algorithm in polynomial time [3] [4]. If it is true then **P = NP**, otherwise, **P** ≠ **NP**. Here **P** denotes the class of decision problems solvable by some algorithm within a number of steps bounded by some fixed polynomial in the length of the input. The notation **NP** stands for "nondeterministic polynomial time" and the class of questions for which an answer can be verified in polynomial time is called **NP**.

The proof of **P = NP** is associated with an important concept, named "**NP**-complete", which was firstly proposed by Cook in 1971 [5]. The **NP**-complete problems are a set of problems to each of which any other **NP**-problem can be reduced in polynomial time, and whose solution may still be verified in polynomial time. That is, any **NP** problem can be transformed into any of the **NP**-complete problems [3] [4]. Hence, to prove the including relationship **NP** $\subseteq$ **P** it only requires the consideration of a single arbitrarily-chosen **NP**-complete problem (**P** $\subseteq$ **NP** is naturally satisfied). The precise expression is given by Cook in [3] as follows:

Proposition 1. *Let L be a language over a finite alphabet. If L is* **NP***-complete and L belongs to* **P** *then* **P = NP**.

It follows from [1] [2] [4] [6] that the travelling salesman problem is **NP**-complete, and *Proposition* 1 indicates that the proof of **P = NP** only requires the seeking of a language L in **P** for it, that is, a fast algorithm which can be executed on computer with a polynomial time rather than an exponential time. Many researchers had already tried on it, and a lot of "optimizing" or "searching" approaches were developed (please see the list in [1]). But the associated computational-complexity problem is still suspended till now. Just as indicated by Devlin in [2], to solve this it requires an unusual but wonderful thinking!

From October 2018, I began to think about this, and a fresh idea suddenly appeared when I deleted the longest one among the total 15 paths (which connect 6 cities separately) by a pen on a paper (see **Figure 1**). Since the candidate for the concerned route is always composed by 6 segments and each one of them has several choices, to shorten the total length the longest path should be avoided. Without path-deleting this problem requires a selection among $5!=5\times4\times3\times2\times1=120$ choices. Yet the deleting of one path implies the avoiding of $4!=24$ choices. Among the left paths, we can continue to delete the maximum one, only if there are more than two paths connecting each city. To repeat this process with 4 times (that is, 4 paths are deleted), the number of the left choices is merely $5!-4\times4!=24$. The effect is very considerable! It

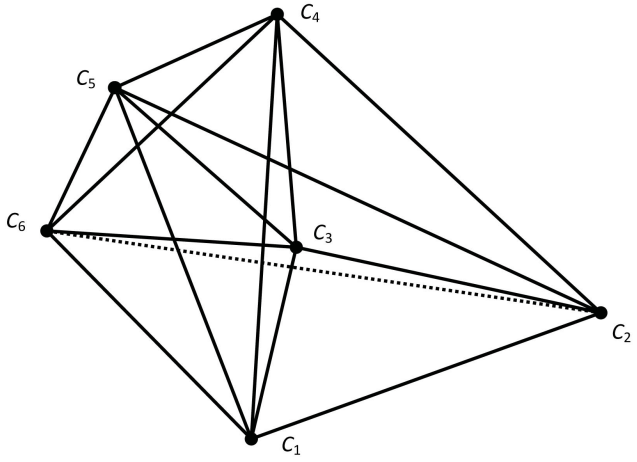

Figure 1. Example for the travelling salesman problem with 6 cites.

may greatly reduce the computational complexity, particularly for the problem with large number of cities. This "maximum-deleting method" had thrown some lights on the travelling salesman problem and the **P** versus **NP** problem. The subsequent endeavor on the preciseness lifted the mysterious veils of them. The present paper is a report on these.

2. Maximum-Deleting Method

There is a necessity to re-express the travelling salesman problem in mathematical language. Let n be the total number of the concerned cities (include the origin city, $n \geq 4$), which are expressed as a series of nodes C_k specified by the two-dimensional coordinates (x_k, y_k) $(1 \leq k \leq n)$ (the order of the nodes is arbitrarily chosen). The path between every pair of cities is abstracted as a line-segment (in the following we call it by "line" for short) and the one between the i-th and the j-th notes is denoted by $l_{i,j}$ $(1 \leq i, j \leq n$ with $i \neq j)$. Notice that $l_{j,i}$ and $l_{i,j}$ denote the same line, only the case $i < j$ is considered in the following. All these lines compose a set:

$$D^* = \left\{ l_{i,j} : 1 \leq i, j \leq n, i < j \right\},$$

whose number of elements is

$$N = (n-1) + (n-2) + \cdots + 2 + 1 = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}. \tag{1}$$

The concerned route that visits each city and returns to the origin city can be expressed as a closed loop Q which connects all the n nodes with two requirements:

1) *Each node has two connections* (*that is, it only connects two lines*);

2) *The journey length of* Q (*that is, the sum of the lengthes for the selected* n *lines in* D^*) *should be shortest among all the choices.*

Under the frame of maximum-deleting method, the first requirement is a criterion and the second requirement is a strategy.

To demonstrate the strategy in a clear way, we arrange the elements in D^* by comparing their lengthes defined by the Euclidean distance

$$\left| l_{i,j} \right| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2}$$

and get

$$D_0 = \left\{ l_{p(k),q(k)} : 1 \leq k \leq N, \left| l_{p(r),q(r)} \right| \geq \left| l_{p(r+1),q(r+1)} \right| \text{ for all } 1 \leq r \leq N-1 \right\},$$

where N is given in Equation (1) and the subscripts $p(k)$ and $q(k)$ are two mappings. For example, in case $l_{2,5}$ has the biggest length in D^* then it reads $l_{p(1),q(1)}$ in D_0. For a candidate of Q, its journey-length reads

$$S = s_1 + s_2 + \cdots + s_n = \sum_{k=1}^{n} s_k, \tag{3}$$

where the chosen lines are also arranged according to the length, and s_k denotes the length of the k-th line with $s_k \geq s_{k+1}$ for all $1 \leq k \leq n-1$. To put

the connectivity aside, the upper and lower bounds of S accord with the choice of the first n terms and the last n terms in D_0, respectively. Since the candidate of Q is always composed by n segments and each one of them has many choices, to shorten the total length the longest line should be avoided. Precisely, if the length s_1 in Equation (3) corresponds to the first line $l_{p(1),q(1)}$ in D_0, then the substitution of this line by another one in D_0 with a shorter length s_1' should be beneficial for shortening the possible length S. After deleting $l_{p(1),q(1)}$ one can continue to delete $l_{p(2),q(2)}$, $l_{p(3),q(3)}$, ⋯, only if the line to be deleted has more than two connections on each endpoint. *This is an ensemble compressing strategy which squeezes the candidate set for Q close to the last n terms of* D_0. General speaking, it seldom occurs for the last n terms composing a single closed loop (the lower bound of S is seldom achieved). So there always exists some lines $l_{p(k),q(k)}$ with $k \le N-n$ in the left set, that is, some shorter lines are deleted. This leaves a certain leeway for modifications in the last process where the shortest principle should be obeyed. We note that, when the deleting process is finished, the left lines usually compose many small loops which need to be connected into a single one.

3. Fast Algorithm with Polynomial Time

Notice that requirement (I) for Q only needs 2 of the $n-1$ (≥ 3) connections for each node, it is always possible for executing the maximum-deleting strategy. The maximum-deleting algorithm for the travelling salesman problem is as follows:

Step 1. To input n nodes C_k and generate all the lines in D^*.

Step 2. To calculate the corresponding line lengths by Equation (2) and arrange them in the form of D_0 [the mapping from D^* to D_0 needs to be saved].

Step 3. To delete the lines in D_0 one by one from the beginning (see **Figure 1**), only if the line to be deleted has more than two connections on each endpoint. If there are only two connections left on one of the two endpoints, then skip this line and continue to delete the subsequent smaller one[1].

Step 4. If there are some particular nodes, to transform them into the normal ones. For the case in **Figure 2**, for example, one can delete $l_{t,r}$ (that is, the line between the nodes C_t and C_r) and $l_{t,s}$ and substituted them by adding a single line $l_{r,s}$. To follow the shortest principle, it needs to compare the lengthes of $|l_{r,s}|+|l_{t,p}|+|l_{t,q}|$, $|l_{s,p}|+|l_{t,q}|+|l_{t,r}|$, $|l_{p,q}|+|l_{t,r}|+|l_{t,s}|$ and $|l_{q,r}|+|l_{t,s}|+|l_{t,p}|$ and select the shortest one to substitute. For the case in **Figure 3**, the comparison is done on the last three lengthes. For example, in case $|l_{q,r}|+|l_{t,s}|+|l_{t,p}|$ is the shortest, one can delete $l_{t,q}$ and $l_{t,r}$ and add $l_{q,r}$. If there are some

[1]After the deleting process, except some particular nodes with $2m$ ($m \ge 2$) connections, all the others are the normal nodes which have 2 connections. The reason is that, originally all the nodes have the same number of connections and all the deleted lines connect pairs of nodes, it is impossible for a sole node who remains odd number of connections. In addition, a particular node only connects the normal nodes in its neighborhood, since if it connects another particular one at least two lines between them should be deleted.

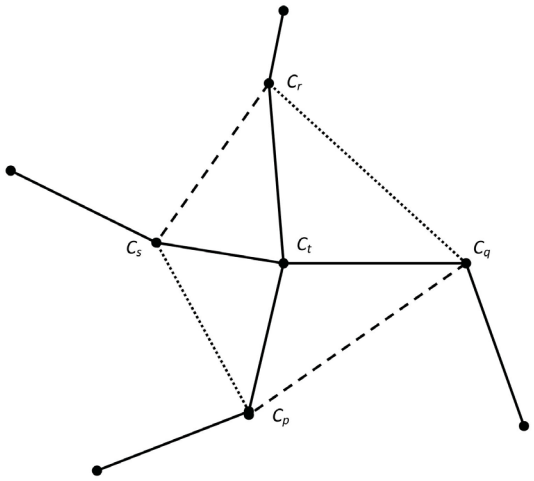

Figure 2. One case with particular node.

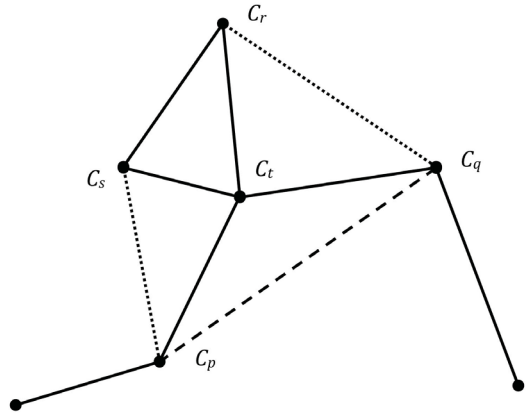

Figure 3. Another case with particular node.

particular nodes who own $2m$ ($m \geq 3$) connections, one can repeat this processing and delete the lines pair by pair until only a pair of connections are left[2].

Step 5. If there are two crossed lines as in **Figure 4** (responding to a twisted loop), to substitute them by the shortest pair of opposite sides of the quadrilateral[3]. That is, if $|l_{p,q}|+|l_{r,s}|<|l_{s,p}|+|l_{q,r}|$, then delete $l_{s,q}$ and $l_{p,r}$ and add $l_{p,q}$ and $l_{r,s}$. For the case in **Figure 5**, to delete $l_{s,q}$, $l_{p,r}$ and add $l_{q,r}$, $l_{s,p}$ in a direct way. Other crossed cases can be dealt with in the same way[4].

Step 6. If the left n lines compose many isolated closed loops, to connect them in the following way: For the cases in **Figure 6** and **Figure 7**, to find the nearest two neighboring pair of points and make the substitution. Specifically, if $|l_{s,p}|+|l_{r,q}|$ is the shortest length among all the choices, then delete $l_{s,r}$, $l_{p,q}$ and add $l_{s,p}$, $l_{r,q}$.

[2]After Step 4 all the left nodes are the normal ones and the number of left lines is n.

[3]$|l_{s,q}|+|l_{p,r}|=OC_s+OC_r+OC_p+OC_q>|l_{p,q}|+|l_{r,s}|,|l_{s,p}|+|l_{q,r}|$.

[4]After Step 5 The left n lines compose either a single closed loop or some isolated closed loops without twists.

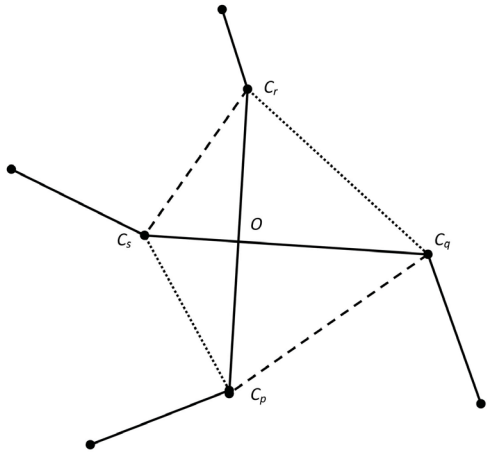

Figure 4. A case with crossed lines.

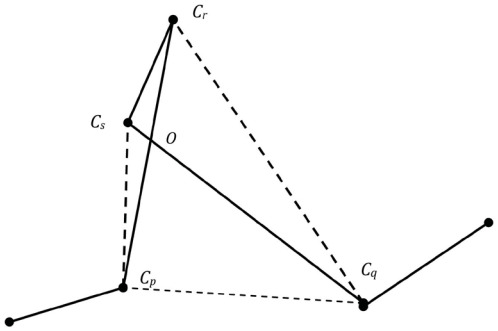

Figure 5. Another case with crossed lines.

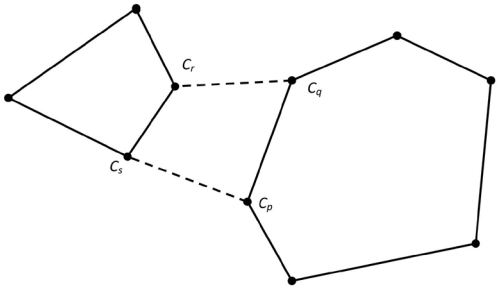

Figure 6. One positional relation for two isolated loops.

Step 7. To modify the unique closed loop according the shortest principle. Firstly, to re-number the nodes from C_1 along this loop in an anticlockwise order; Secondly, to follow this order and search from C_1 for the triangle determined by three neighboring nodes who owns a shortest boundary which is not on the loop, and then make the possible substitution. For example, in **Figure 8** the triangle determined by C_k, C_{k+1} and C_{k+2} ($k \geq 1$) has a shortest boundary $l_{k,k+2}$ which is not on the loop. If $|l_{k,k+2}|+|l_{k+1,k+3}|<|l_{k,k+1}|+|l_{k+2,k+3}|$, then delete $l_{k,k+1}$, $l_{k+2,k+3}$ and add $l_{k,k+2}$, $l_{k+1,k+3}$. To continue searching for this kind of triangles along the loop and do the processing one by one until it returns back to C_1.

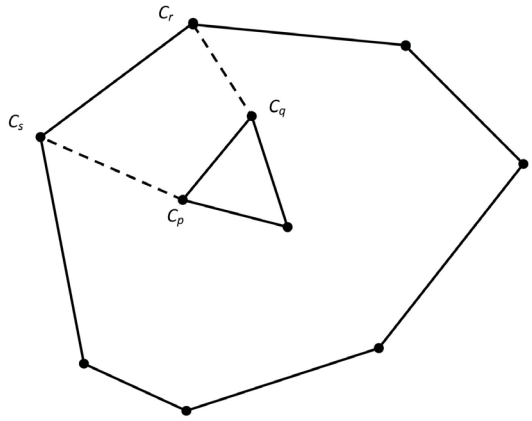

Figure 7. Another positional relation for two isolated loops.

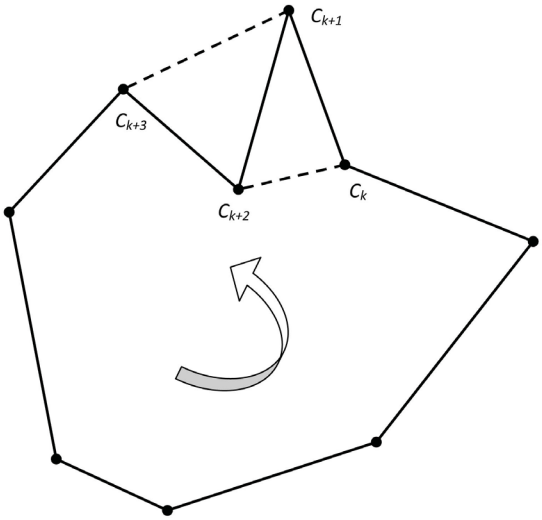

Figure 8. The possible case in the final modifying process.

According to this algorithm, when Step 3 is finished all the possible longer lines are deleted and the left ones compose either a single closed loop or many small closed loops. The determined thing is that the firstly generated normal node only remains two shortest connections. For the subsequent generated normal nodes, in the sake of connectivity each of them either remains two shortest connections or share one or two lines with the previously generated nodes (it doesn't matter whether they connect a particular node). Since the deleting processing is done from the longest one, it reduces all the possible longer connections respect to all the n nodes, and, to insure the connectivity, there are no other choices for a given node possessing a relatively shorter connection. So the closed loop to be found is based on the left lines with the shortest sum of lengths. In the subsequent steps, the small loops are opened and connected together. During this process the shortest principle is obeyed and the total length of the candidate loop is reduced to the maximum extent. So the final closed loop has the shortest length among all the choices. It is the anticipated solution for the travelling salesman problem.

Theorem 1. *The maximum-deleting algorithm for the travelling salesman problem has a polynomial time of order* $O(n^4)$.

Proof: It follows from the definition of D^* that the first step of this algorithm costs computation time of about $K_1 N = K_1 \times n(n-1)/2 = O(n^2)$, where K_1 is a constant. To select the longest line in D^* it needs to make $N-1$ times comparison. To select the second longest line it needs to make $N-2$ times comparison $\cdots$. So the arranging process for all the lines in the second step needs a computation time:

$$\sum_{k=1}^{N-1} k = \frac{N(N-1)}{2} = \frac{1}{8} n^2 (n-1)^2 - \frac{1}{4} n(n-1) = O(n^4).$$

The corresponding saving process needs a time of about $O(N) = O(n^2)$. The deleting process in Step 3 also needs a time of about $O(N) = O(n^2)$. From Step 4 to Step 7, only some of the lines are adjusted. In each step, the number of involved lines is no more than $2n$. To include the judgements and substitutions, all the calculating process costs computation time of about $O(n)$. Hence, the computational complexity of this algorithm is determined by the second step, and all the process requires a polynomial time of order $O(n^4)$. The proof is finished.

Corollary 1. P = NP.

Proof: On the one hand, it follows from [1] [2] [4] [6] that the travelling salesman problem is **NP**-complete; On the other hand, we have found a fast algorithm for the travelling salesman problem which can be executed on computer with a polynomial time. In another word, we have found a language L for a **NP**-complete problem in **P**. It follows from *Proposition* 1 that **P = NP**. The proof is finished.

4. Conclusion

By using a new approach named "maximum-deleting method", we have constructed a fast algorithm for the travelling salesman problem with a polynomial time of order $O(n^4)$, which will greatly reduce the computational complexity. Notice that this problem is **NP**-complete, as a corollary, we have also solved the well-known open problem named "**P** versus **NP**". The result indicates that **P = NP** which will result in a surprise to all, since great majority of people have believed that **P** ≠ **NP**.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Wikipedia (the Free Encyclopedia), Travelling Salesman Problem. https://en.wikipedia.org/wiki/Travelling_salesman_problem

[2] Devlin, K.J. (2003) The Millennium Problems: The Seven Greatest Unsolved Ma-

thematical Puzzles of Our Time. Basic Books, New York.

[3] Cook, S. (2018) Official Problem Description: The P versus NP Problem. http://www.claymath.org/millennium-problems/p-vs-np-problem

[4] Wikipedia (the Free Encyclopedia), P versus NP Problem. https://en.wikipedia.org/wiki/P_versus_NP_problem

[5] Cook, S. (1971) The Complexity of Theorem-Proving Procedures. *Conference Record of Third Annual ACM Symposium on Theory of Computing*, ACM, New York, 151-158.

[6] Garey, M.R. and Johnson, D.S. (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco.