Polynomial-Space Exact Algorithms for Traveling Salesman Problem in Degree Bounded Graphs

Norhazwani Md Yunos

Polynomial-Space Exact Algorithms for Traveling Salesman Problem in Degree Bounded Graphs

Norhazwani Md Yunos

Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University
Kyoto, Japan

February 2017

Doctoral dissertation
submitted to the Graduate School of Informatics, Kyoto University
in partial fulfillment of the requirement for the degree of
DOCTOR OF INFORMATICS
(Applied Mathematics and Physics)

Preface

The Traveling Salesman Problem, TSP for short, is one of the most well-known NP-hard optimization problems, and it has been extensively studied in various fields of optimization. It has been formulated as a mathematical problem in the 1930s, and many algorithmic methods have been investigated to address the challenge of finding the fastest algorithm in terms of the running time. A recent trend of research focuses on trying to alleviate the time and space complexity of algorithms for solving the TSP by focusing on special types of TSP instances, namely graphs of limited degree. Let degree-i graph stand for a graph in which vertices have at most i incident edges. In this thesis, we design a series of polynomial-space branching algorithms for the TSP in degree bounded graphs, namely the TSP in degree-5, degree-6, degree-7 and degree-8 graphs. More specifically, this thesis shows that the TSP in graphs with maximum degree 5 can be solved in $O^*(2.4723^n)$, the TSP in graphs with maximum degree 6 can be solved in $O^*(3.0335^n)$, the TSP in graphs with maximum degree 7 can be solved in $O^*(3.5939^n)$, and the TSP in graphs with maximum degree 8 can be solved in $O^*(4.1577^n)$. To the best of our knowledge, each of the algorithms proposed in this thesis is the first exact algorithm specialized to graphs of such high degree.

All these algorithms employ similar techniques as most of the previous branching algorithms for the TSP, where the idea behind the branching algorithm is to solve subproblems recursively by using two sets of rules, namely reduction rules and branching rules. For the reduction rules, we use simple natural observations. For the branching rules, we introduce a set of branching rules for each of the algorithms to perform the branching operation. The nature of our branching operation is to branch on an unforced edge e iteratively, by either including edge e into a solution or excluding edge e from all solutions. The choice of edge e to branch on plays a key role in the analysis of our branching algorithm. To this effect, in this thesis we have assigned a way on how to choose the edge e to branch on. In the analysis of the running time, we use the measure-and-conquer method as a tool to get an upper bound of the running time.

As a result, the presented polynomial-space branching algorithms for the TSP in degree-5, degree-6 and degree-7 graphs outperform the time complexity of the algorithm for the TSP in a general n-vertex graph of Gurevich and Shelah's $O^*(4^n n^{\log n})$ (SIAM Journal of Computation, 16(3), pp 486–502, 1987). On the other hand, the time complexity of the algorithm for the TSP in degree-8 graphs has breached the $O^*(4^n n^{\log n})$-time

algorithm due to Gurevich and Shelah. This answers the question which is the highest degree i such that our approach for designing and analyzing algorithms specialized to the TSP in degree-i graphs has a lower bound on the time complexity than the algorithm due to Gurevich and Shelah.

We believe that our algorithms are significant and will give some contributions towards practical applications, such as routing and scheduling problems, and possibly beyond. It is our hope that the work done can be serve as a basis for future advancement in related topics.

Norhazwani Md Yunos
February 2017
Kyoto, Japan

Acknowledgement

All praise for God who is the most Gracious, most Compassionate. This PhD thesis may never seen the light without the help of many generous people.

First and foremost, my sincerest appreciation must go to my supervisor, *Professor Dr. Hiroshi Nagamochi*, who took the risk of supervising me even knowing that I was from slightly different background. Many thanks for his brilliance, guidance, advice, patience and constant care; which I am grateful to have you as the supervisor and it will be a priceless experience that I will never forget.

I am sincere grateful to *Dr. Aleksandar Shurbevski* in addition to providing invaluable academic support and guidance from the inception till the successful completion of the research. I benefited greatly from many fruitful discussions and I cannot forget each of the valuable help and motivation he gave to me. I must observe that he also went largely out of his way to make my transition to a new environment seamless and my continued stay as pleasant as possible.

I am also thankful to *Professor Dr. Yoshito Ohta* of Kyoto University, as well as *Professor Dr. Nobuo Yamashita* of Kyoto University, for serving on my dissertation committee. I would like to extend my thanks to my lab-mates, the members of the Discrete Mathematics Laboratory, for their help and supported in any respect during the course of the research, especially to *Shahrizan*, *Fei He*, *Ken Iwaide* and *Yuhei Nishiyama*.

Special thanks go to my dearest husband, *Hasrul Nisham Rosly*, for his patience, love, constant support and understanding through thick and thin. Even though we live apart during my studies, but he always be there for every step of this process, supported me, listened to me, calmed me down, and above all, he believed that I would accomplish my life goal. You are everything to me, my life partner, my good friend, as well as my best quarrel-mates. Your subsistence for traveling Malaysia-Japan once every two months is greatly appreciated. My loving thanks go to my precious princess, *Zara Elviana Hasrul Nisham*, thank you so much for always stand by my side and accompanying me all the good and bad times throughout this memorable journey. Although you might not understand my situation and what I am doing, you are always my everything. Being a full-time mom as well as playing a role as a dad to a child who is actively growing up and need full attention and in the same time trying to juggle full-time studies, while the immediate family members were more than 5000 kilometers apart could be really overwhelming. But

praise to the Almighty for the health favors especially to both my daughter and me, I am survived staying here without any big obstacle. For my husband and my daughter, your wife/mom have to struggle for my life goal and thank you so much for all your patience, supports and understandings.

A million thanks goes to my beloved dad, *Md Yunos Hasan*, and my beloved mom, *Halimahtun Mahphoz*, for their prayers, love, constant support and frequents commute Malaysia-Japan for accompanying and motivating me when I am down. Both of you are the source of endless selfless love and you always encourage me in any endeavor in my life. As time goes by, I remember when I first left to Japan, they were always stationed by their phones, ready to console me when I was feeling homesick. It would not be possible to be the person I am today if it were not from the way you upbringing me. For the best education you had given to me and for all kinds of sacrifices, I dedicate this higher certificate specially for you two! Besides, I also would like to express my million thanks to my father-in-law, *Rosly Pin*, and my mother-in-law, *Noor Al-Huda Abd Rahman*, for their prayers, affection and their understanding for leaving their son while I were away as full-time PhD students.

A great thanks to all my siblings, you guys are really supportive especially at the end of the journey when I am madly homesick. Also thank you so much to all my family and friends in Malaysia, as well as my ex-lecturers, who always send their regards and always keep me motivated by virtue of modern communications technology. I also feel thankful for the support from my fellow post-graduates, with whom we share our struggle stories, you know who you are.

The most important thankful and acknowledgement must go to my scholarship, *Ministry of Higher Education (MoHE) Malaysia* and *University Teknikal Malaysia Melaka (UTeM)*. Without this, I will never be able to set my feet on this top 100 university in the world and second top ranking university in Japan. To all Malaysians, I am humbled by your money from which the scholarship is originated. I am now just completed this historical and memorable journey and ready to go back to serve my nation. Rest assured, your money would not go to waste. Now it is time for me to leave Japan for good after 3 years undertaking this PhD research.

Last but not least, to those who have made contributions directly or indirectly and cannot all be named, thank you very much.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Optimization Problem

As time goes by, a lot of problems arise in various areas of study, such as in economics, engineering and natural science. For example, the necessity of dealing with various organizational and planning problems often makes use of several analysis techniques in mathematics. Such problems occur when a decision maker must make a decision in order to manage a system with some specific criteria in an optimal way. We call such problems *optimization problems*. Some of the first mathematicians to manoeuvre optimization problems were Fermat, Euler, several members of the Bernoulli family, Lagrange, and others in connection with the development of Calculus in the 17^{th} and 18^{th} centuries [23].

An optimization problem can be easily described as a problem with a collection of variables or instances that determine a collection of solutions, and requests to find the best solution among them. Particularly, there are two important terms in an optimization problem, an *objective function* and a *feasible region*. The feasible region is a set of all solutions, and a solution in the feasible region is called *feasible*. In deriving a solution, the objective function requests to find the *optimal* value over all feasible solutions. In an optimization problem, the word "optimal" usually refers to *minimum* or *maximum*, where a minimization (resp., maximization) problem asks to minimize (resp., maximize) the objective function. For example, a single-variable minimization problem can be described by

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to:} \quad & x \in X, \end{aligned}$$

where f is a given objective function, $x \in \mathbb{R}^n$ is a decision variable in n-dimensional real vector, $X \subseteq \mathbb{R}^n$ is a given feasible region, $\mathbb{R}$ is the set of real numbers, and $\mathbb{R}^n$ is the n-dimensional vector space over $\mathbb{R}$. A feasible solution x^* is optimal to the optimization

problem if and only if $f(x^*) \leq f(x)$ holds for all feasible solutions $x \in X$.

There are two categories of optimization problems, *continuous optimization problems* and *combinatorial optimization problems*. If a problem has a continuous feasible region such as a set of real numbers or a function, then it is called a continuous optimization problem. If a problem has a discrete feasible region such as a set of integers, permutations or graphs, then it is called a combinatorial optimization problem.

Linear Programming, LP for short, and *Integer Linear Programming*, ILP for short, are mathematical techniques to solve optimization problems. The history behind the LP formulation goes back to 1939, when it was discovered by Leonid Kantorovich [44]. He had developed an LP formulation to solve optimization problems during the World War II on how to plan expenditures and returns to reduce the costs of the army and the losses incurred by the enemy. Since then, LP and ILP have been used widely to solve many optimization problems. For instance, in organization and planning management, a decision maker has to make the most effective use of an organization's resources such as labor, money, time and raw material. This is to guarantee that the products such as clothing, food, furniture and electrical devices, or services such as airline schedule and investment policies, can be produced in an optimal way. One of the successful optimization problems that use LP and ILP as their solution method is scheduling school buses, where the problem asks to minimize to total distance traveled when carrying students.

A variety of continuous and combinatorial optimization problems appear in many real world problems and as a consequence many algorithms for their solution have been proposed. The topic of this thesis is categorized under combinatorial optimization problems.

1.2 Algorithm

An *algorithm* is a problem-solving method that has been widely used in computational problems. It takes some value or a set of values as an *input* and produces some value or set of values as an *output*. The input to an algorithm is called an *instance* of the problem, and the size of the input of the algorithm is referred to the size of the instance. Thus, an algorithm that solves a problem is a step-by-step procedure to solve a given problem instance. Precisely, an algorithm is a set of instructions that transform the input into an output. An algorithm is said to be *correct* if for every input instance, it terminates with a correct output [14].

For example, let us take the sorting problem, a problem where we need to sort a set of numbers into a non-decreasing order. The input and the output of the problem are defined by:

Input: A set $\{a_1, a_2, \ldots, a_n\}$ of n numbers.

Output: A list $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

For instance, given the input set $\{23, 20, 85, 58, 11\}$, a sorting algorithm returns as output the set $\langle 11, 20, 23, 58, 85 \rangle$, as shown in Figure 1.1. An instance of the sorting problem is a set of n numbers, and the size of the input of the sorting algorithm is n.

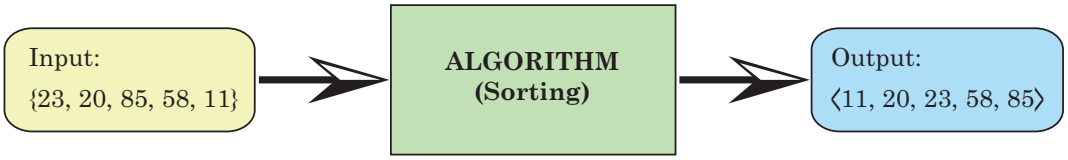

Figure 1.1: An overview of an algorithm.

In the analysis of algorithms, in addition to the correctness of algorithms, the performance of algorithms is also important. One of the reasons why the analysis of an algorithm is performed is to compare different algorithms for the same task. The performance of algorithms is measured by their running time. The running time of an algorithm is calculated in terms of fundamental mathematical quantities by doing a mathematical analysis on the quantities involved [45]. Usually, the running time of an algorithm is defined to be the number of elementary steps for the algorithm to execute in order to deliver its output and it is stated as a function relating the size of the input instance to the numbers of steps, known as *time complexity*.

There are various attributes of an algorithm's running time, as shown in Figure 1.2. Some algorithms may run faster on certain data sets than others. Thus, finding an *average case* can be very difficult, and hence the *worst-case* running time is measured. However, in certain applications such as air traffic control and surgery, knowing the worst-case running time is not important, but finding the *best-case* running time is the matter. Worst case (resp., best case) running time, known as *upper running time bound* (resp., *lower running time bound*) is denoted by $\mathcal{O}$ (resp., Ω) with respect to a function relating to the size of the input instance, to tell the maximum (resp., minimum) steps of the algorithm to give its output.

In the analysis of the running time of algorithms, the time complexity is stated as a function with respect to *the size of the input instance*. Basically, it tells how fast a function grows or declines. Henceforth, the running time of an algorithm is only considered up to the leading term of a function, such as cn^2, and ignoring the constant coefficient of the leading term, c, because the smaller-order terms of a function and the coefficient of the leading term are less significant for large values of n. For example, if the running time $T(n)$ of an algorithm is given by $2n^2+n-1$, then the upper time bound of the algorithm is at the order of n^2, and we write $T(n) = \mathcal{O}(n^2)$. Usually we do not know the exact running

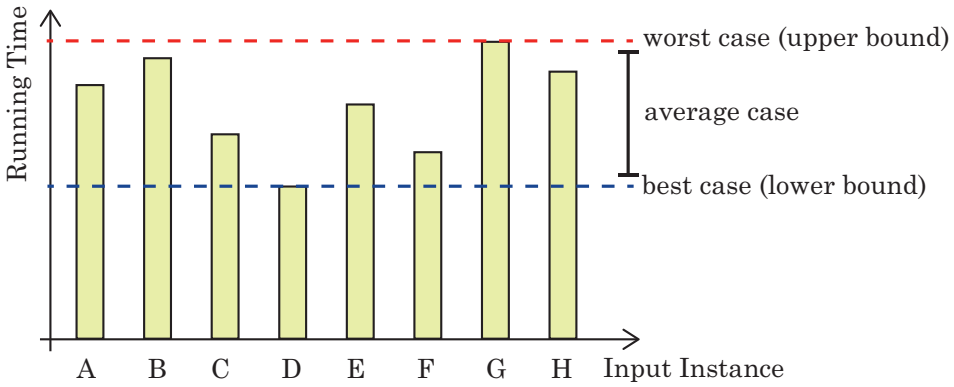

Figure 1.2: Variety attribute of the algorithm's running time.

time $T(n)$, and we derive only an upper bound on $T(n)$ in the form of $T(n) = \mathcal{O}(f(n))$.

There are numerous algorithms for numerous problems in the world, and undoubtedly different algorithms give different time and space complexities. The characterization of which is an *efficient* algorithm always depends on the situation. However, computer scientists recognized a simple characterization that we can consider to differentiate algorithms based on their time complexity. They are largely classified as *polynomial-time* algorithms and *exponential-time* algorithms [21]. Thus, if the running time of an algorithm is bounded by a polynomial of an input instance size, then the algorithm is considered as efficient.

In the area of theoretical computer science, exact algorithms are designed so that upper bounds on their worst case time complexity can be theoretically analyzed as a function of the input size. On the other hand, many existing solvers, for example IBM ILOG CPLEX Optimization Studio (CPLEX), routinely used in practice run sufficiently fast by relying on heuristics and bounding operations whose worst or average time complexities are difficult to be analysed theoretically. On a set of instances, an exact algorithm with a low theoretically obtained worst time complexity may still not run as fast as a practical solver. However, it has been recently reported by Akiba and Iwata [1] that some exact algorithms designed to improve theoretical time bounds do run sufficiently fast as compared with solvers developed for solving instances practically. Thus, it is utterly important to continue theoretical research and to develop algorithms with ever lower bounds on their computational complexity, as these can show to be highly relevant in practice as well. Therefore, research on theoretical algorithms are also important and significant since it has been proven by Akiba and Iwata [1] that some theoretical algorithms run sufficiently fast as compared to practical solvers.

1.3 Computational Complexity

Computational complexity theory is one of the major branches of study in theoretical computing science and mathematics. Basically, a *computational problem* is a problem

where we are given an input and we want to return an output that satisfies some properties. We can classify computational problems in two classes, namely $\mathcal{P}$ and $\mathcal{NP}$. We denote by $\mathcal{P}$ the class of problems solvable in polynomial time, and by $\mathcal{NP}$ the class of problems that admit a nondeterministic polynomial time algorithm. We call a problem $\mathcal{NP}$-hard, if the polynomial solvability of the problem would imply that all other problems in $\mathcal{NP}$ are solvable in polynomial time as well. We call a problem $\mathcal{NP}$-complete, if the problem is in the class $\mathcal{NP}$ and is $\mathcal{NP}$-hard [30].

The concept of $\mathcal{NP}$-completeness was introduced by Stephen Cook in 1971 [12]. Since then, $\mathcal{NP}$-completeness is the cornerstone of complexity theory. Until now, determining whether the class $\mathcal{P}$ and the class $\mathcal{NP}$ are the same or not is still a major open question, in other words, whether $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$. The question whether $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$ is one of the seven millenium problems [13].

Assuming that $\mathcal{P} \neq \mathcal{NP}$, the relation of $\mathcal{NP}$-complete problems are shown in Figure 1.3. Dasgupta et al. [16] have classified some problems according to their classes, as shown in Table 1.1. On the left-hand side of the table, there are some examples of problems in $\mathcal{P}$ that can be solved by diverse specialized algorithms, such as dynamic programming or greedy algorithms. Whereas in the right-hand side of the table, there are some $\mathcal{NP}$-complete problems that have escaped efficient solution over many decades or centuries.

As we can see from Table 1.1, the *Traveling Salesman Problem* is one of the $\mathcal{NP}$-complete problems. When the theory of $\mathcal{NP}$-completeness was developed, the Traveling Salesman Problem was one of the first problem to be proven as an $\mathcal{NP}$-hard problem by Karp in 1972 [29].

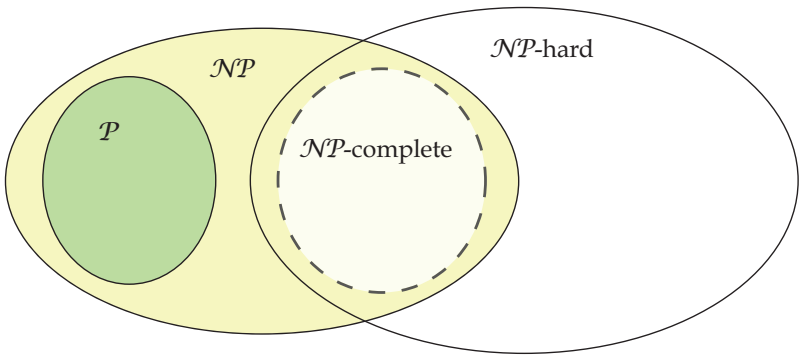

Figure 1.3: Classes of computational problems.

1.4 The Traveling Salesman Problem

The Traveling Salesman Problem, or TSP for short, gained much attention and has been studied by researchers from many areas such as mathematics, computer science and oper-

Table 1.1: List of problems in their class, assuming that $\mathcal{P} \neq \mathcal{NP}$.

Problems in $\mathcal{P}$	$\mathcal{NP}$-complete Problems
Linear Programming	Integer Linear Programming
Shortest Path	Longest Path
Minimum Spanning Tree	Traveling Salesman Problem
2-Satisfiability (2-SAT)	3-Satisfiability (3-SAT)
Bipartite Matching	3D Matching
Minimum Cut	Balanced Cut

ations research. There is a long and great history behind the birth of the TSP as written in the book of Cook [13]. Basically, the wave of the TSP started in the 1930s by Merrill Flood who stimulated the interest of TSP in many quarters, and one of them in obtaining near optimal solutions in reference to routing of school buses [13].

The TSP is a problem where we are given the distances between each pair of n cities, and we need to visit every city exactly once and return to the home city, with a minimal cost of travelling. In practice, it is very easy to describe, but it is very difficult to solve efficiently. As the number of cities increases, the determination of the optimal tour becomes incredibly complex.

1.4.1 Applications of the TSP

The TSP is one of the most extensively studied problems in any field of optimization, and has been used as a framework to solve other problems. In other words, the TSP can be applied to solve many problems by reducing them to a TSP formulation. For example take the plotter, a computer printer for printing vector graphics which uses a pen to draw pictures on paper. The TSP can be applied as a procedure to direct the movement of the pen while drawing, so that useless moves are avoided, and the pen travels a minimal distance.

There are a variety of problems that can be solved using TSP formulations. One of the widely adopted direct applications of the TSP is in drilling problems of printed circuit boards, PCBs, as reported in Grotschel et al. [24]. This drilling problem asks to position the drilling head where holes have to be drilled through the board, while the holes may be of different sizes. An illustration of a PCB with different sizes of holes is shown in Figure 1.4. To drill two holes of different diameters consecutively, the drilling head of the machine has to move to a tool box and change the drilling equipment. This is quite time consuming, and thus one has to choose one diameter and drill all holes of the same diameter, and later, change to other size of diameter and drill the holes of the