The Hong Kong Polytechnic University
Department of Logistics and Maritime Studies

Approximability of Vehicle Routing Problems

by

XU Liang

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
September 2011

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Singed)

_____(Name of student)

ABSTRACT

Approximability of Vehicle Routing Problems

by

Liang Xu

Vehicle Routing Problems (VRPs) can be described as a class of combinatorial optimization problems that seek to determine a set of feasible routes for a fleet of vehicles to serve customers at different locations, so as to optimize certain objective functions. Due to the growth of the transportation and logistics industries, many new VRPs are emerging in different contexts. Since they are usually *NP*-hard, these problems call for the design of approximation algorithms that achieve constant approximation ratios. Moreover, in the existing literature there are several VRPs whose constant-ratio approximation algorithms are either unknown or improvable. Therefore, in this thesis we study the approximability of the following three categories of VRPs, by either developing the constant-ratio approximation algorithms or deriving approximation hardness results.

The first category of problems includes the min-max Path Cover Problem (PCP) and its variants, which aim to determine a set of k paths for k vehicles to serve customers in a metric undirected graph, so that the maximum total edge and vertex weight among all paths is minimized. This category of problems was introduced in the literature in the context of "vehicle routing for relief efforts". Since they

are relatively new in the literature, approximation algorithms are almost unknown. We consider four variants of the min-max PCP, in which the vehicles have either unlimited or limited capacities, and they start from either a given depot or any depot of a given depot set. We have developed approximation algorithms for these variants, which achieve approximation ratios of $\max\{3-2/k, 2\}$, 5, $\max\{5-2/k, 4\}$, and 7, respectively. For these four variants of PCP, we have also proved their first approximation hardness results, by showing that unless *P=NP*, it is impossible for them to achieve approximation ratios less than 4/3, 3/2, 3/2, and 2, respectively.

The second category of problems includes the min-max k-Traveling Salesmen Problem on a Tree (k-TSPT) and its variants. With k a given positive integer denoting the number of salesmen, which is independent of the input size, the min-max k-TSPT aims to determine a set of k tours for the k salesmen to serve all the customers that are located in a tree, such that the k tours all start from and return to the depot, so as to minimize the maximum length of the k tours. In the literature, the question as to whether or not the min-max 2-TSPT yields a pseudo-polynomial time exact algorithm has remained open for a decade. We have provided a positive answer to this open question by developing a pseudo-polynomial time exact algorithm using a dynamic programming approach. Based on this dynamic program, we have further developed a fully polynomial time approximation scheme for the problem. Moreover, we have generalized these algorithms for the min-max k-TSPT for any given constant $k \geq 2$, and we have extended them to other variants.

The third category of problems includes the k-depot Traveling Salesmen Problem (k-depot TSP) and its variants. The k-depot TSP is an extension of the single-depot TSP, and it aims to determine a set of k tours for the k vehicles to serve all the customers on a metric undirected graph so that the total length of the tours is minimized. We have shown that a non-trivial extension of the well-known Christofides' heuristic has a tight approximation ratio of $2-1/k$, which is better than the existing

2-approximation algorithm available in the literature. This result is significant when k is small. Moreover, we have demonstrated how this algorithm can be applied to the development of approximation algorithms for other multiple-depot VRPs.

ACKNOWLEDGEMENTS

I would like first of all to thank The Hong Kong Polytechnic University for offering me the opportunity to pursue my Ph.D degree. The research works presented in this thesis were supported in part by Hong Kong Research Grant Council (RGC), General Research Fund (GRF), No. Poly U 532010.

I also wish to extend my sincere gratitude to my chief supervisor, Dr Zhou Xu, for his instructive advice and useful suggestions. Intellectually, I have benefited greatly from his deep insights and research ideas in completing my thesis. His acute sense of research and his attitude towards it have impressed me. From him, I have learnt the necessary expertise and the skills needed to conduct qualified research. Although I know I have quite a long way to go, Dr Xu has prepared me mentally for the challenges in my future academic career.

During my three-year Ph.D study, I was very fortunate to have Professor Chung-lun Li as my co-supervisor. He was always more than willing to share with me his wealth of knowledge and expertise. Most of all, being able to cooperate with Professor Li in developing an academic paper has been an unforgettable experience. I will fondly remember the efficiency of our joint work and the spirit of team-working he instilled in me that benefit me throughout my life.

My thanks also go to my co-author and friend, Dr Dongsheng Xu, as well as the anonymous referees of Naval Research Logistics and Operations Research Letters, whose insightful suggestions and useful comments helped me to considerately improve the content of this thesis.

I would also like to thank my great teachers at PolyU in my Ph.D study, Dr Pengfei Guo, Dr Xiaowen Fu, Dr Li Jiang, and many others. Their delightful willingness to share has equipped me with necessary knowledge and experience to help tackle challenges, not only during my Ph.D study, but also in my future academic career. Moreover, I am also indebted to those teachers outside PolyU, Professor Liu Hong, Dr Man-Cho So, and Dr Nan Chen, for allowing me to sit as an audience in their classes, and always treating me with patience.

I am also thankful for my two close friends and research colleagues, Xiaofan Lai and Libing Yang, whose willingness to share both their knowledge in mathematical programming and expertise in coding has been of immense help.

I also want to express my appreciation to my thesis committee members, Dr Daniel Ng, Dr Karthik Natarajan, and Professor May-Yee Leung for their time spent on my thesis, and their valuable suggestions and comments.

Finally, I would like to express my heartfelt gratitude to my family members. Without their support and patience, I can hardly imagine making it through my study and research smoothly.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

CHAPTER 1

Introduction

1.1 Background

Vehicle Routing Problems (VRPs) can be described as a class of combinatorial optimization problems that seek to determine a set of feasible routes for a fleet of vehicles to serve customers at different locations, so as to optimize certain objective functions. The VRPs have wide applications in the transportation and logistics industries. Among many others, typical applications include solid waste collection, street cleaning, school bus routing, bank crew scheduling, and salesmen routing [75].

Due to different motivations, there are many variants of the VRPs that have been studied in previous literature. One of the most classic VRPs is the Traveling Salesman Problem (TSP), which aims to determine the shortest close route in a weighted graph for a single vehicle to visit each customer exactly once. Most of the VRPs are *NP*-hard since they can be reduced from the Hamiltonian cycle problem or the Hamiltonian path problem, which aim to determine whether there exists a simple cycle or path that visits each vertex of a given graph exactly once, and both of them are known to be *NP*-complete. For example, the TSP is *NP*-hard even when the graph is undirected and the edge weights of the graph form a metric that is symmetric and satisfies the triangle inequality, because it can be reduced from the Hamilton cycle problem by setting the distance between two vertices to 1 if they are adjacent, and to 2 otherwise.

For such VRPs, which are *NP*-hard, it seems unlikely that one can develop exact algorithms that can determine optimal solutions in polynomial time. Thus, it is of great interest to know whether or not one can develop approximation algorithms to achieve certain worst-case performance guarantees. Here, a ρ-approximation algorithm is defined as a polynomial time algorithm that produces a feasible solution of objective value no more than ρ times the objective value of an optimal solution to a minimization problem [35]. The value of ρ indicates the worst-case performance guarantee of the approximation algorithm, and is therefore called an approximation ratio. A family of ρ-approximation algorithms is a fully polynomial time approximation scheme (FPTAS) if $\rho = 1 + \epsilon$ for any $\epsilon > 0$ and its running time is polynomial in $1/\epsilon$. Moreover, for a minimization problem, if one can show that there exists a constant θ such that for any $\rho < \theta$, no ρ-approximation algorithm exists unless *P=NP*, then the value of θ indicates the approximation hardness of the problem, and is called an inapproximability bound [35].

Due to the growth of the transportation and logistics industries, many new VRPs are emerging in different contexts. Since they are also *NP*-hard, these problems call for the design of approximation algorithms that can achieve constant approximation ratios, and call for the study on inapproximability bounds. Moreover, in the existing literature, there are several VRPs, whose constant ratio approximation algorithms are either unknown or expected to be further improved. Therefore, in this thesis, we study the approximability of the following three categories of VRPs, by either developing the first or improved constant ratio approximation algorithms, or deriving inapproximability bounds for approximation hardness results.

The first category of problems includes the min-max Path Cover Problem (PCP) and its variants, which are to determine a set of paths for vehicles to serve customers in a metric undirected graph, so that the maximum total edge and vertex weight among all paths is minimized. The min-max PCP has appeared in the literature

recently in the context of "routing for relief efforts", where the minimization of latest delivery time becomes critical so that the deliveries of humanitarian aid are both fast and fair [10]. Like other VRPs, the min-max PCP has different variants restricted to different constraints, such as the vehicle capacity constraint, the vehicle depot restriction, etc. The min-max PCP is *NP*-hard, because it can be reduced from the Hamiltonian path problem by setting the number of vehicles to be one, and setting the distance between two vertices to 1 if they are adjacent, and to 2 otherwise. However, the study of approximation algorithms and inapproximability bounds are limited for the min-max PCP and its variants. Therefore, it is of great interest to investigate the following research question:

Q1: To what extent can one develop constant ratio approximation algorithms for the min-max PCP and its variants?

The second category of problems studied in this thesis includes the min-max k-Traveling Salesmen Problem on a Tree (k-TSPT) and its variants. The min-max k-TSPT aims to determine a set of k tours for the k vehicles to serve all the customers that are located in a tree, such that the k tours all start from and return to the depot, so as to minimize the maximum length of the k tours. Like other VRPs, the min-max k-TSPT has different variants restricted by different constraints or under different objectives. This part of the thesis follows a large body of research on VRPs with customers located in a tree-shaped network, which appears in many practical situations, such as railway systems for pit mines [54], rural delivery systems with roads that branch off from a single highway [11, 76], and water systems with a main stream and several tributaries [51].

When $k = 1$, the min-max k-TSPT is equivalent to the TSP on a tree, which can be solved in polynomial time by simply visiting each edge of the tree twice. When k is a given constant greater than two, the min-max k-TSPT is *NP*-hard [8]. It is known only that the min-max 2-TSPT has a 4/3-approximation algorithm [8].

Accordingly, Averbakh and Berman [8] raised an open question on whether or not the min-max 2-TSPT has a pseudo-polynomial time exact algorithm. Initiated by this open question, we investigate the following research question for the min-max k-TSPT and its variants:

Q2: Can one develop an exact algorithm that yields a pseudo-polynomial running time for the min-max k-TSPT or its variants, for any given constant $k \geq 2$? Can one develop an FPTAS for the min-max k-TSPT or its variants, for any given constant $k \geq 2$?

The third category of problems studied by this thesis includes the k-depot TSP and its variants. The k-depot TSP is an extension of the single depot TSP, and aims to determine a set of tours for k vehicles from distinct depots to serve all the customers on a metric undirected graph, so as to minimize the total length of the tours. In its typical applications, vehicles are dispatched from a number of different depots. For example, in the dairy industry, road tankers based at various locations are used to collect and transport milk from farms to processing plants [32, 39]. In military applications, unmanned aerial vehicles situated at different bases are routed to sites to execute various missions [16, 17, 69].

The k-depot TSP for $k \geq 2$ is *NP*-hard since it can be directly reduced from the TSP. For the TSP on a metric undirected graph, it is known that a tree algorithm achieves an approximation ratio of 2 [64, 73], and that a heuristic given by Christofides [19] achieves an approximation ratio of 3/2. Thus, the best approximation ratio that can be expected for the k-depot TSP is 3/2 unless the TSP on a metric undirected graph has an approximation algorithm superior to the Christofides' heuristic. However, the best available approximation ratio for the k-depot TSP in the literature is 2, which is achieved by an approximation algorithm that is based on the tree algorithm [69]. In the literature [3, 59, 69], it has also been suggested that the Christofides' heuristic can be extended for the k-depot TSP. However, worst-case

analysis of the extended heuristic is known to be difficult [3, 59]. It therefore remains an open question if a tight approximation ratio of the extended Christofides' heuristic can be found for the k-depot TSP. To resolve this open question, we investigate the following research question in this thesis:

Q3: What is the tight approximation ratio of the extended Christofides' heuristic for the k-depot TSP?

1.2 Literature Review

Since the first VRP model, namely, the truck dispatching problem, was first introduced by Dantzig and Ramser [23], VRPs have been studied extensively in the literature. Due to the *NP*-hardness of the VRPs, it is unlikely that one can develop an exact algorithm that can solve all the instances of a vehicle routing problem into optimality in polynomial time. Thus, a great deal of research has been devoted to developing exact algorithms to solve restricted instances into optimality, to designing heuristic algorithms to produce near-optimal solutions for randomly generated or practical instances, and to developing approximation algorithms with constant approximation ratios for worst-case guarantees. The book edited by Toth and Vigo [75] surveys exact algorithms proposed for the VRPs up to 2002. Laporte and Semet [55] and Gendreau et al. [37] have reviewed various heuristics provided for the VRPs in the literature. More recent heuristics for the VRPs can be found in the survey of Cordeau et al. [22]. Concerning the approximation algorithms for the VRPs, the survey by Arora [4] can be referred for more details.

As with other classic combinatorial optimization problems, exact algorithms for the VRPs rely mostly on a branch-and-bound procedure [1, 20, 29], which derives lower bounds and upper bounds on the optimal objective values of sub-problems to fathom nodes in the branch-and-bound tree, so as to obtain a global optimal solution.

There are two typical variants of the branch-and-bound algorithm for the VRPs. One is called the branch-and-cut [6, 30, 57], which, at each node in the branch-and-bound tree, iteratively refines the linear relaxation of the integer programming formulation of the problem by adding valid linear inequalities, so as to obtain relatively tighter lower bounds. The other is called the branch-and-price [18, 24, 72], which, at each node in the branch-and-bound tree, uses a column generation method to obtain relatively tighter lower bounds.

The family of heuristics for the VRPs can be broadly categorized into two classes: classic heuristics and meta-heuristics. The classic heuristics are usually based on constructive and improvement procedures [13, 31, 38], while the meta-heuristics center on a thorough exploration of most of the potential regions of solution space [36]. The solutions produced by the meta-heuristics often have better quality than the solutions obtained by the classic heuristics, while the price to pay for the meta-heuristics is the increased running time.

This thesis follows a large volume of literature on the development of constant ratio approximation algorithms for VRPs. In the remainder of this section, we are going to review the relevant works on the approximability for the three categories of VRPs studied in this thesis. Like many other works in the literature, we assume that the edge weights of the underlying graph of the VRPs form a metric that is symmetric and satisfies the triangle inequality, because the vehicles can always travel along the shortest path between any two vertices.

1.2.1 Relevant works for the min-max PCP and its variants

There is a growing body of literature on vehicle routing problems under the min-max objective. Most of them focus on the min-max cycle cover problems and the min-max tree cover problems, in which customers are serviced by a set of k cycles and a set of k trees, respectively. Frederickson et al. [34] have proposed an approx-

imation algorithm for the min-max k-traveling salesmen problem (min-max k-TSP) that is equivalent to the min-max cycle cover problem with a single depot. Their algorithm first applies the classical Christofides' [19] heuristic to obtain an approximate cycle that covers all customers, and then splits the cycle almost evenly into k segments. By joining the depot to both the endpoints of each segment, a cycle cover with an approximation ratio of $(5/2 - 1/k)$ is obtained. Bazgan et al. [12] have analyzed the approximation hardness of the min-max k-TSP. Averbakh and Berman [9] and Nagamochi and Okada [62] have developed approximation algorithms with error bounds for the min-max k-TSP on a tree. For min-max tree covers, Arkin et al. [3] have developed a 4-approximation algorithm for the min-max k-tree cover problem with no root given. Even et al. [28] have achieved the same approximation ratio for the problem with a given set of roots. When only one root is given, Nagamochi [61] has improved the approximation ratio to $(3 - 2/(k + 1))$. When the given graph is a tree and a root is given, Nagamochi and Okada [63] have improved the approximation ratio to $(2 + \epsilon)$, where ϵ is a positive constant which may be set arbitrarily close to zero. Xu and Wen [79] have analyzed the approximation hardness of several variants of min-max tree cover.

There are a few results available on various min-max path cover problems, but they do not take the service handling time into consideration. Campbell et al. [15] have analyzed the worst-case performance of the solution that minimizes the total travel time under the min-max objective. They have shown that for a special case of the min-max uncapacitated path cover with a single depot where the service handling time is ignored, the latest service completion time of the solution that minimizes the total travel time is no greater than $2k$ times the minimum latest service completion time. Hoogeveen [44] developed a 3/2-approximation algorithm for the traveling salesmen path problem, which is equivalent to the min-max uncapacitated path cover problem with a single vehicle. When replenishment is not allowed after the vehicles have left

their starting points, so that each vehicle can service at most Q customers, Campbell et al. [15] have developed a heuristic but without any constant approximation ratio guarantee. Arkin et al. [3] have devised a 4-approximation algorithm for the problem with no service handling time, no depot, and uncapacitated vehicles.

Although service handling time has been ignored by most of the related works in the literature, it often contributes to a significant portion of the latest service completion time and should be included as vertex weights, i.e., $h(v)$ for all $v \in V$, in the problem formulation. Moreover, addressing vertex weights for the min-max path cover problem has significant research value, due to the challenges explained as follows: Consider the traditional vehicle routing problems of minimizing the total edge and vertex weight of all the vehicle routes. Such an objective is a sum of two separated parts, including the total edge weight and the total vertex weight. Since the second part always equals $\sum_{v \in V} h(v)$, which is a constant, the vertex weights in these problems can be ignored [75]. However, this separation property does not hold when we have a min-max objective, and therefore the vertex weights cannot simply be ignored in various min-max path cover problems. Note that even under the min-max objective, some vehicle routing models seek for tours instead of paths. When tours are sought, one can always revise the edge weights as $\widetilde{w}(u,v) = w(u,v) + h(u)/2 + h(v)/2$ for each $(u,v) \in E$ to include the vertex weights. The revised edge weights, $\widetilde{w}$, still form a metric, and the total edge weight of every cycle under $\widetilde{w}$ equals its total edge and vertex weight under w and h [75]. However, this property of $\widetilde{w}$ is not valid in the min-max path cover problem, because the latest service completion time of a path may not include the time spent on the return trip from its last customer to a depot. Therefore, it is of great research interest to develop constant ratio approximation algorithms for various min-max path cover problems, with the service handling time taken into consideration.

When customers are located on a path or tree, several constant ratio approxima-

tion algorithms are known for an extension of the min-max uncapacitated path cover problem, namely, the vehicle scheduling problem (VSP), where the task to be served for each customer has a release time and a processing time. For the VSP on a path with a single vehicle, Karuno et al. [50] presented a 2-approximation algorithm, which was later extended and improved by Karuno and Nagamochi [47] and Karuno and Nagamochi [48], even for the multiple-vehicle case. For the VSP on a tree, Hu [45] developed a 3-approximation algorithm, and Karuno and Nagamochi [48] presented a polynomial time approximation scheme that achieves an approximation ratio of $(1+\epsilon)$ for any $\epsilon > 0$, with the running time polynomial in the input size but exponential to $1/\epsilon$. However, for the VSP on a metric undirected graph, no constant ratio approximation algorithms are known in the literature.

Finally, in contrast to algorithm designs, results on approximation hardness are almost unknown for various min-max path cover problems. It is only known that the asymmetric TSP cannot be approximated with a ratio less than 117/116 unless P=NP, whereas the symmetric TSP cannot be approximated with a ratio less than 220/219 [65].

1.2.2 Relevant works for the min-max k-TSP on a tree and its variants

There is a large body of literature on VRPs on trees, because the tree-shaped networks appear in many practical situations for vehicle routing, such as in railway systems for pit mines [54], in production lines [14, 49], and in rural delivery systems with roads that branch off from a single highway [11, 76]. In some water systems, such as the Pearl River Delta in China, the main stream and its tributaries often form a tree-shaped or even a line-shaped network [51, 66].

Given that the network is tree-shaped, many VRPs are known to have better approximation algorithms or even polynomial time exact algorithms. For example, when $k = 1$, the min-max k-TSPT is equivalent to the TSP on a tree, which can be

solved in polynomial time by visiting each edge of the tree exactly twice, while the general version of the TSP is known to be *NP*-hard. For the min-max 2-TSP, the best existing approximation algorithm achieves an approximation ratio of 2 [12], while for the min-max 2-TSPT, Averbakh and Berman [8] obtained a 4/3-approximation algorithm. For any given constant $k \geq 2$, although the k-TSPT is known to be *NP*-hard, it still remains an open question on whether or not a pseudo-polynomial time exact algorithm or an FPTAS can be developed for the min-max k-TSPT.

There are several studies in the literature on approximation algorithms for different variants of the min-max k-TSPT. For example, Averbakh and Berman [8] developed a 3/2-approximation algorithm for the min-max 2-depot TSPT, which is an extension of the min-max 2-TSPT by allowing vehicles to start from different depots. Averbakh and Berman [9] studied another variant of the min-max k-TSPT, which is called the min-max k-TSPT location-allocation problem, where one needs to assign a home location (or depot) to each vehicle. For this variant, Averbakh and Berman [9] developed a $(k-1)/(k+1)$-approximation algorithm, whose running time is $O(k^{k-1}n^{k-1})$, which is polynomial time if k is a given constant. Nagamochi and Okada [62] later achieved the same approximation ratio but reduced the running time to $O((k-1)!n)$, which, however, is still exponential to k. For a special case of the min-max k-TSPT, where only optimal home locations must be found, without the corresponding tours, Averbakh and Berman [10] presented polynomial time exact algorithms for those special cases with $k = 2$ and $k = 3$. However, as far as we know, no pseudo-polynomial time exact algorithms are known for these variants of the min-max k-TSPT for any given constant $k \geq 2$.

Besides the min-max objective, other objectives have also been studied in the literature on approximation algorithms for the VRPs on a tree. Averbakh and Berman [7] studied a VRP on a tree with the objective of minimizing the total waiting time of the customers, and showed that the problem is polynomial solvable for the single-

vehicle case, and is strongly *NP*-hard for the multiple-vehicle case. Karuno et al. [49] studied a VRP on a tree with the objective of minimizing the maximum lateness of tasks served for the customers subject to their release times. It was shown that the single vehicle case of the problem is strongly *NP*-hard in general, but is polynomial solvable if only depth-first routing is allowed for the vehicle. Moreover, several authors studied a capacitated vehicle routing problem on a tree (CVRPT), which aims to determine a set of routes for vehicles from a given depot to deliver goods to satisfy demands of customers on a tree, such that for each vehicle, the number of goods delivered by it cannot exceed a specific capacity limit, so as to minimize the total edge weight of the routes. When the demand of each customer is not splittable, Labbé et al. [54] developed a 2-approximation algorithm for the CVRPT. When the demand of each customer is splittable, Hamaguchi and Katoh [42] developed a 3/2-approximation algorithm, which was later improved by Asano et al. [5] to achieve an approximation ratio of 1.3078. Recently, for the CVRPT with multiple depots, where each given depot has a vehicle with a capacity, Hu [45] developed a 2-approximation algorithm.

In addition to the VRPs, the tree-shaped network has also been considered for other combinatorial optimization problems. For example, Johnson and Niemi [46] introduced a tree knapsack problem, which can be regarded as a 0-1 knapsack problem on a rooted tree such that if a vertex is selected, then all the vertices on the path from the selected vertex to the root must also be selected. The objective of the tree knapsack problem is to minimize the total weight of the edges that join the selected vertices. Based on the fact that each edge of the tree can be contained at most once in any feasible solution, Johnson and Niemi [46] developed a dynamic programming algorithm that returns an optimal solution in pseudo-polynomial time for any given instance of the tree knapsack problem. Similarly, for other variants of the tree knapsack problem, such as the tree partitioning problem [53, 58, 77] and the

tree facility problem [74], one can also obtain pseudo-polynomial time algorithms by a similar dynamic programming approach. However, for the k-TSPT with a given constant $k \geq 2$, since each edge of the tree may be traversed by more than one vehicle, the dynamic programming approach for the tree knapsack problem cannot be directly applied.

1.2.3 Relevant works for the k-depot TSP and its variants

The k-depot TSP for $k \geq 2$ is *NP*-hard since it can be reduced from the single-depot TSP on a metric undirected graph by setting the depot set D to consist of the single-depot of the TSP and taking the $(k-1)$ additional depots to have infinite distances from the customers. This motivates the development of approximation algorithms that can give good solutions in polynomial time and which guarantee constant approximation ratios. Indeed, a number of researchers have developed approximation algorithms with constant approximation ratios for the TSP on a metric undirected graph [19, 73] and other single-depot vehicle routing problems [26]. These are typically based on two methods: A tree algorithm [64, 73] and a heuristic given by Christofides [19]. Both approaches use a basic framework comprising the following three steps: Step 1: Find a minimum spanning tree (MST) of the given graph; Step 2: Create a connected Eulerian multigraph by adding a set of edges to the MST; Step 3: Find a Eulerian closed walk of the multigraph, remove repeated vertices of the closed walk, and return the resulting cycle.

The two approaches differ in step 2, where the tree algorithm duplicates all edges of the MST obtained in step 1, whereas the Christofides' heuristic adds to the MST only edges of a minimum perfect matching for vertices having odd degrees in the MST. Both guarantee that every vertex is connected and has even degree so that there exists a Eulerian closed walk in the multigraph. As the MST is not longer than the optimal TSP tour, the tree algorithm has an approximation ratio of 2. The

Christofides' heuristic, however, achieves a superior ratio of 3/2, since the minimum perfect matching used in its step 2 is not longer than half of the optimal TSP tour.

In contrast with single-depot vehicle routing problems, many previous works on multiple-depot vehicle routing problems do not provide worst-case analysis [39, 71, 82]. Based on the tree algorithm, Rathinam et al. [69] developed a 2-approximation algorithm for the k-depot TSP which is the best available in the literature. Malik et al. [59] has provided a similar 2-approximation algorithm for a generalized k-depot TSP, in which more than k depots with distinct salesmen are available in D, of which only k can be selected to serve customers. Moreover, for some extensions of the k-depot TSP, such as the capacitated case of the k-depot TSP [43, 56], where each vehicle can serve a limited number of customers, and the generalized k-depot TSP [59], in which more than k traveling salesmen are available but only k of them can be selected to service the customers, although constant ratio approximation algorithms are known, the approximation ratios are all greater than or equal to 2. Since the best approximation ratio of 3/2 for the TSP is obtained using the Christofides' heuristic, the best ratio that can be expected for the k-depot TSP is 3/2, unless the TSP on a metric undirected graph has an approximation algorithm superior to the Christofides' heuristic.

In the literature [3, 59, 69], it has been suggested that the Christofides' heuristic can be extended for the k-depot TSP and its variants by computing a constrained spanning forest in which each tree contains a distinct depot in step 1 above. However, the worst-case analysis of the extended Christofides' heuristic is known to be difficult since this requires bounding the length of a minimum perfect matching for vertices having odd degrees in the constrained spanning forest [3, 59]. Rathinama and Sengupta [70] recently extended the Christofides' heuristic to obtain a 3/2-approximation algorithm for a 2-depot Hamiltonian path problem, which determines paths instead of tours for salesmen. The analysis of the approximation ratio in their work is

manageable since it only requires bounding the length of a partial matching for a 2-depot case. Moreover, two recent technical reports [67, 68] claimed to achieve approximation ratios better than 2 for two multiple-depot Hamiltonian path problems. However, we note that the proofs for Proposition V.2 in [68] and Proposition II.3 in [67] assume that any concave-convex function f of vectors π and x must satisfy $f(\pi^*, x^*) = \max_\pi f(\pi, x^*)$ for all (π^*, x^*) with $f(\pi^*, x^*) = \max_\pi \min_x f(\pi, x)$, which is not always the case, in particular for the (π^*, x^*) constructed in [67, 68]. Since these methods that were developed in the mentioned technical reports seem to be incorrect, they are not expected to be applied to the k-depot TSP to address the open question.

It therefore remains an open question if a tight approximation ratio of the extended Christofides' heuristic can be found for the k-depot TSP. Moreover, besides the TSP, many researchers have studied approximation algorithms for other single-depot vehicle routing problems on a metric undirected graph, such as the clustered TSP [2, 40], the mixed Chinese postman problem [21, 25, 26, 33], the rural postman problem [27, 34], and the stacker-crane problem [27, 34]. For these problems, the best approximation algorithms available are all dependant on approximation solutions to the TSP [26, 27]. As far as we are aware, multiple-depot extensions of these more complex routing problems have rarely been studied in the approximation algorithm literature. Therefore, it is possible to extend our proposed approximation algorithm for the k-depot TSP to develop constant ratio algorithms for these previously mentioned multiple-depot extensions.

1.3 Contributions, Organization and Publications

This thesis contributes to the study on approximability for three categories of VRPs, namely, the min-max PCP, the min-max k-TSPT, and the k-depot TSP, by investigating the three research questions, Q1, Q2 and Q3, proposed in Section 1.1. With new techniques developed, we are able to obtain improved constant-ratio ap-

proximation algorithms and/or the first inapproximability bounds for these VRPs and their variants. As a result, the two open questions mentioned in Section 1.1 have been settled. The main results in this thesis have been summarized in Table 1.1, where the numbers in brackets indicate the best known results in the literature:

Table 1.1: Summary of main results in this thesis.

Problems	Approximation Ratios	Inapproximability Bounds
Min-max uncapacitated path cover problem with single depot	$\max\{3-2/k, 2\}$	4/3
Min-max uncapacitated path cover problem with multiple depots	5	3/2
Min-max capacitated path cover problem with single depot	$\max\{5-2/k, 4\}$	3/2
Min-max capacitated path cover problem with multiple depots	7	2
Min-max 2-traveling salesmen problem on a tree	$1+\epsilon$ (4/3)	
Min-sum k-depot traveling salesmen problem	$2-1/k$ (2)	

Regarding the research question Q1, we have developed the first approximation algorithms and the first inapproximability bounds for two cases of the min-max PCP respectively, one for the uncapacitated case where no vehicle has capacity limit, and the other for the capacitated case where customers served by each vehicle cannot exceed a capacity limit. For the min-max Uncapacitated PCP (UPCP), we have obtained the following results for the development of approximation algorithms, as shown in Chapter 2:

1. We have developed a cycle-splitting-by-weight procedure, which has been used extensively in the development of constant ratio approximation algorithms for the min-max UPCP.

2. For the min-max UPCP with a single depot available for all the vehicles, we

have developed the first constant ratio approximation algorithm, which achieves an approximation ratio of $\max\{3 - 2/k, 2\}$.

3. For the min-max UPCP with multiple depots available for all the vehicles, we have developed the first constant ratio approximation algorithm, which achieves an approximation ratio of 5.

For the min-max Capacitated PCP (CPCP), we have obtained the following results for the development of approximation algorithms, as shown in Chapter 3:

1. We have developed a cycle-splitting-by-customer-number procedure, which has been used extensively in the development of constant ratio approximation algorithms for the min-max CPCP.

2. For the min-max CPCP with a single depot available for all the vehicles, we have developed the first constant ratio approximation algorithm, which achieves an approximation ratio of $\max\{5 - 2/k, 4\}$.

3. For the min-max CPCP with multiple depots available for all the vehicles, we have developed the first constant ratio approximation algorithm, which achieves an approximation ratio of 7.

In addition, we have also demonstrated the possible extensions of our techniques, so as to develop constant ratio approximation algorithms for other variants of the min-max PCP. For example, in Section 3.6.1, we have illustrated that the technique developed for the min-max path cover problem with single depot can be extended to the min-max cycle cover problem with single depot. The approximation algorithm developed for the min-max cycle cover problem with single depot has a superior approximation ratio than the min-max path cover problem with single depot.

For both the min-max UPCP and the min-max CPCP, we have obtained the first inapproximability bounds in Chapter 4, by reductions from the 3-Dimensional Matching Problem (3DM) [35], which is a well-known *NP*-complete problem.

1. For the the min-max UPCP with a single depot for all the vehicles, we have shown that it is impossible to achieve an approximation ratio less than 4/3, unless *P=NP*.

2. For the the min-max UPCP with multiple depots for all the vehicles, we have shown that it is impossible to achieve an approximation ratio less than 3/2, unless *P=NP*.

3. For the the min-max CPCP with a single depot for all the vehicles, we have shown that it is impossible to achieve an approximation ratio less than 3/2, unless *P=NP*.

4. For the the min-max CPCP with multiple depots for all the vehicles, we have shown that it is impossible to achieve an approximation ratio less than 2, unless *P=NP*.

In addition, we have also demonstrated the possible extensions of our reductions, so as to derive inapproximability bounds for other variants of the min-max PCP.

Regarding the research question Q2, we have provided a complete answer in Chapter 6:

1. We have developed an exact algorithm for the min-max 2-TSPT by a dynamic programming approach, which yields a pseudo-polynomial running time. This also provides a positive answer to the open question raised by Averbakh and Berman [8].

2. Based on the pseudo-polynomial time exact algorithm, we have developed an FPTAS for the min-max 2-TSPT.

3. We have shown that both the exact algorithm and the FPTAS can be extended for the min-max k-TSPT for any given constant $k \geq 2$.

In addition, we have also demonstrated the possible extensions of our algorithms for several other VRPs on a tree-shaped network.

Regarding the research question Q3, we have provided a complete answer in Chapter 6:

1. We have developed a set of new upper bounds on the length of the matching used in the extended Christofides' heuristic for the k-depot TSP. The new upper bounds are extensively used in deriving the approximation ratio of the extended Christofides' heuristic.

2. We have showed that the extended Christofides' heuristic is a $(2-1/k)$-approximation algorithm. This approximation algorithm is better than the 2-approximation algorithm available in the literature, and is close to $3/2$ when k is close to 2.

3. We have devised an example to show that the approximation ratio of the extended Christofides' heuristic is not greater than $(2-1/k)$, which implies that the approximation ratio of $(2-1/k)$ is tight.

In addition, we have demonstrated how the results and the techniques can be generalized for possible improvements to the extended Christofides' heuristic for the k-depot TSP, as well as for the analysis of variants of the extended Christofides' heuristic for other multiple-depot VRPs.

We conclude this thesis in Chapter 7 with discussions on both the limitations and the future research directions.

Finally, some of the results in this thesis have appeared in [78, 80, 81].

CHAPTER 2

Approximation Algorithms for the Min-max Uncapacitated Path Cover Problems

2.1 Introduction

The min-max Path Cover Problem (PCP) aims to determine a set of paths for a given fleet of k vehicles to serve customers in a metric undirected graph, so that the maximum total edge and vertex weight among all paths is minimized. Most of the existing literatures in vehicle routing focus on developing efficient and effective solution methods to find "good" vehicle routes, where the quality of the routes is usually measured in terms of the total travel distance of the vehicles. However, in some applications the minimization of maximum arrival time at the customer locations (i.e., the latest service completion time) is more relevant. Vehicle routing with this type of min-max objective has recently appeared in the literature in the context of "routing for relief efforts", where it is critical that the deliveries of humanitarian aid are both fast and fair [15]. Since the min-max PCP is relatively new, approximation algorithms with constant approximation ratios are almost unknown in the literature for the min-max PCP and its variants.

In this chapter, we develop constant ratio approximation algorithms for two variants of the min-max uncapacitated path cover problem (UPCP) in a metric undi-

rected network, where each vehicle is assumed to have an unlimited capacity. This assumption on uncapacitated vehicles reflects those real-life situations that vehicles deliver services, rather than physical materials, to customers. The remainder of this chapter is organized as follows. Section 2.2 introduces the definition of the min-max UPCP, and the notation that are used throughout this chapter. We then introduce in Section 2.3 a cycle splitting procedure by revised edge weights, which has been extensively applied in the development of approximation algorithms in this chapter. For the min-max UPCP with a single depot available for all the vehicles, we develop in Section 2.4 the first constant-ratio approximation algorithm, which achieves an approximation ratio of $\max\{3 - 2/k, 2\}$. For the min-max UPCP with multiple depots available for all the vehicles, we develop the first constant ratio approximation algorithm in Section 2.5, which achieves an approximation ratio of 5. Finally, we summarize this chapter in Section 2.6.

2.2 Notation and Problem Definition

The min-max UPCP can be described as follows: We are given a complete undirected graph $G = (V, E)$ with vertex set V and edge set E. There are a given set of vehicle depots $D \subset V$, a given set of customer locations $J = V \setminus D$, and $k \geq 1$ identical vehicles each having an infinite capacity $Q = \infty$. Each vertex $v \in V$ is associated with a non-negative vertex weight $h(v)$, which represents the service handling time at the customer location and is equal to zero if v is a depot in D. Each edge $e \in E$ is associated with a non-negative edge weight $w(e)$, which represents the time for the vehicles to travel along the edge. We would like to determine a set of k paths for the k vehicles to service all the customers in J, such that each vehicle must start from a depot in D. The objective is to minimize the latest service completion time among all vehicles, where the service completion time of a vehicle includes all of its edge traveling time and service handling time, but excludes its return trip (if any) from

its last customer back to a depot. We assume that the edge weights form a metric (i.e., they are symmetric and satisfy the triangle inequality), because the vehicles can always travel along the shortest path between any two vertices.

We denote the instance of the uncapacitated min-max path cover problem as (G, D, J, w, h, k, Q). Depending on whether D contains only one depot or multiple depots, the problem has the following two typical variants: the min-max uncapacitated path cover problem with single depot (UPCP-SD), and the min-max uncapacitated path cover problem with multiple depots (UPCP-MD).

Consider any instance $\mathcal{I} = (G, D, J, w, h, k, Q)$ of the min-max uncapacitated path cover problem with an underlying complete undirected graph $G = (V, E)$. A *path* in G is a sequence $P = \langle v_1, v_2, \ldots, v_{m+1} \rangle$ of vertices such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq m$. Vertices v_1 and v_{m+1} are the starting endpoint and the finishing endpoint of P, respectively. If $v_1 = v_{m+1}$, then P is referred to as a *cycle*. The *length* of P, denoted $\ell(P)$, is the number of edges in P (hence $m = \ell(P)$). Thus, if $\ell(P) = 0$, then the path deteriorates to a vertex. We use "$\ell(P) < 0$" to represent the situation where the path is empty. Let $V(P)$, $J(P)$, and $E(P)$ denote the vertex set, customer set, and edge set of P, respectively, where repeated vertices, customers, and edges of P appear only once in these sets. For any subset $U \subseteq V$, we say that the path *covers* U if $U \subseteq V(P)$. Let $w(P) = \sum_{i=1}^{\ell(P)} w(v_i, v_{i+1})$ and $h(P) = \sum_{i=1}^{\ell(P)+1} h(v_i)$ denote the total edge weight and the total vertex weight of P, respectively, where repeated edges and vertices of P are counted repeatedly. Note that if P is a cycle, then in the definition of $h(P)$, the weights of the two endpoints of P are counted twice, even though these two endpoints are actually the same vertex. Of course, if the two endpoints of P are the same depot, then these two endpoints have zero weight. We define a *segment* of path P as a contiguous subsequence of its vertices (which is allowed to be empty).

Consider any set of paths $\mathcal{P}$. Let $\text{cost}(\mathcal{P})$ denote the *cost* of $\mathcal{P}$, which is defined

as the maximum total edge and vertex weight of any path in $\mathcal{P}$:

$$\text{cost}(\mathcal{P}) \quad = \quad \max_{P \in \mathcal{P}} \{w(P) + h(P)\}. \tag{2.1}$$

A *path cover* of $\mathcal{I}$ is a set of paths such that each path in this set starts from a depot, and that each customer of J appears at least once on the paths in this set. A path cover is *exact* if each customer of J appears exactly once among all the paths in the path cover. Thus, a *feasible solution* to $\mathcal{I}$ is an exact path cover that consists of k paths. Because the cost of any feasible solution $\mathcal{P}$ defined in (2.1) is equal to the latest service completion time, an *optimal solution* to $\mathcal{I}$ is a feasible solution that minimizes its cost. We let $\text{OPT}(\mathcal{I})$ denote the cost of an optimal solution of the given instance $\mathcal{I}$. Thus, the min-max path cover problem can be formulated as follows.

Min-max Uncapacitated Path Cover Problem

Instance: $\mathcal{I} = (G, D, J, w, h, k, Q)$, where $G = (V, E)$ is a complete undirected graph, $D \subset V$, $J = V \setminus D$, w is a metric function on E, h is a function on V with $h(d) = 0$ for each $d \in D$, and k is a positive integer and $Q = \infty$.

Feasible solution: An exact path cover $\mathcal{P}$ of $\mathcal{I}$ that consists of k paths.

Objective: $\min_{\mathcal{P}} cost(\mathcal{P})$.

For ease of exposition, we assume that all edge and vertex weights are integers. For any $U \subseteq V$, let $h(U)$ denote the total vertex weight of U. For any $F \subseteq E$, let $w(F)$ denote the total edge weight of F. For any subgraph H of G, let $V(H)$, $J(H)$, and $E(H)$ denote its vertex set, customer set, and edge set, respectively. For each depot $d \in D$, we use $w_{\max}(d) = \max_{v \in J} w(d, v)$ to denote the largest weight among those edges joining d and the customers in J, which equals zero if J is empty. We use $h_{\max} = \max_{v \in V} h(v)$ to denote the largest weight among the vertices in V. For

any $v \in V$, Let $w_{\min}(v, D) = \min_{d \in D} w(v, d)$ denote the smallest weight among those edges connecting v and the depots in D.

Finally, we define a *revised edge weight* for each $(u, v) \in E$ as $w(u, v)+h(u)+h(v)$, which is denoted by $w'(u, v)$. Accordingly, we use $w'(F)$ to represent the total revised edge weight of any $F \subseteq E$. We use $w'(P)$ to represent the total revised edge weight of any path P, where repeated edges are counted repeatedly. Note that if P contains only single vertex, then by definition, $w'(P) = 0$. Lemma 2.1 provides an important property of function w'.

Lemma 2.1. *Consider any cycle C with at least one edge but no repeated customers. It satisfies $w'(C) = w(C) + 2h(J(C))$. Furthermore, for each segment S of C, if S contains no edge, then $w(S) + h(J(S)) \leq h_{\max}$; otherwise, $w(S) + h(J(S)) \leq w'(S)$.*

Proof. Denote $C = \langle v_1, v_2, \ldots, v_{m+1} \rangle$, where $v_1 = v_{m+1}$ and $m = \ell(C) \geq 1$. By definition of w',

$$\begin{aligned} w'(C) &= \sum_{i=1}^{m} w'(v_i, v_{i+1}) = \sum_{i=1}^{m} \big[w(v_i, v_{i+1}) + h(v_i) + h(v_{i+1})\big] \\ &= w(C) + \big[h(C) - h(v_{m+1})\big] + \big[h(C) - h(v_1)\big]. \end{aligned} \tag{2.2}$$

Because $v_1 = v_{m+1}$ and $\langle v_1, v_2, \ldots, v_m \rangle$ has no repeated customers, we have $h(C) = h(J(C)) + h(v_{m+1}) = h(J(C)) + h(v_1)$. Thus, from (2.2) we obtain $w'(C) = w(C) + 2h(J(C))$.

For each segment S of C, if S contains no edge, then $w(S)+h(J(S)) = h(J(S)) \leq h_{\max}$. Otherwise, let v_p and v_q denote the two endpoints of S, where $1 \leq p < q \leq m+1$. By definition of w',

$$w'(S) = w(S) + \big[h(S) - h(v_p)\big] + \big[h(S) - h(v_q)\big]. \tag{2.3}$$

Since $h(S) \geq h(v_p) + h(v_q)$ and $h(S) \geq h(J(S))$, we obtain $w'(S) \geq w(S) + h(J(S))$.

□

2.3 Cycle Splitting by Revised Edge Weights

Approximation algorithms developed in this chapter rely on a cycle splitting procedure, which splits a cycle into a set of segments, such that the revised edge weights of the segments do not exceed some thresholds.

The cycle splitting procedure is denoted by Split$(C, f, \{b_i\}_{1\le i\le f})$. Given a cycle $C = \langle v_1, v_2, \ldots, v_{\ell(C)+1}\rangle$ with no repeated customers (except maybe at its endpoints), a positive integer f, and non-negative thresholds $b_1, b_2, \ldots, b_f$ satisfying $\sum_{i=1}^{f} b_i = w'(C)$, this procedure splits cycle C into f segments $S_1, S_2, \ldots, S_f$ such that $w'(S_i) \le b_i$. A description of this cycle splitting procedure is given below:

Algorithm 2.1 (Split$(C, f, \{b_i\}_{1\le i\le f})$)**.**
Input: A cycle $C = \langle v_1, v_2, \ldots, v_{\ell(C)+1}\rangle$ with no repeated customers (except maybe at its endpoints), an integer $f \ge 1$, and non-negative thresholds $b_1, b_2, \ldots, b_f$ with $\sum_{i=1}^{f} b_i = w'(C)$.
Output: A set $\mathcal{S} = \{S_i : 1 \le i \le f\}$ of f segments of C with $w'(S_i) \le b_i$.

1. Set $\pi(0) := 0$ and $t := 0$. If $w'(C) \le b_1$, then return $\mathcal{S} = \{S_i : 1 \le i \le f\}$, where $S_1 = \langle v_1, v_2, \ldots, v_{\ell(C)}\rangle$ and $S_i = \langle v_1\rangle$ for $2 \le i \le f$. Otherwise, go to Step 2.

2. Repeat the following until $t = f$ or $\pi(t) = \ell(C) + 1$:

   (a) Increment t by 1. Determine $\pi(t)$ as follows:

      i. If $t = f$ or $\pi(t-1) + 1 = \ell(C) + 1$, then set $\pi(t) := \ell(C) + 1$.

      ii. Otherwise, if $w'(v_{\pi(t-1)+1}, v_{\pi(t-1)+2}) > b_t$, then set $\pi(t) := \pi(t-1) + 1$, else set $\pi(t)$ equal to the largest index in $\{\pi(t-1)+2, \pi(t-1)+3, \ldots, \ell(C)+1\}$ such that $w'(\langle v_{\pi(t-1)+1}, v_{\pi(t-1)+2}, \ldots, v_{\pi(t)}\rangle) \le b_t$.

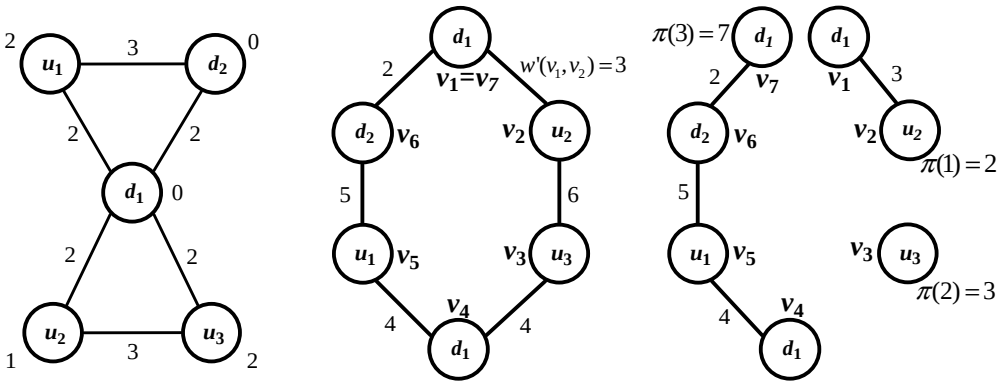
(b) Set segment $S_t := \langle v_{\pi(t-1)+1}, v_{\pi(t-1)+2}, \ldots, v_{\pi(t)} \rangle$.

3. Suppose the iterations in step 2 stop at $t = \tau$. Then, set $S_i := \langle v_1 \rangle$ for $\tau + 1 \leq i \leq f$. Return $\mathcal{S} = \{S_i : 1 \leq i \leq f\}$. □

If $w'(C) \leq b_1$, then Algorithm 2.1 terminates in step 1, and it returns the segment $\langle v_1, v_2, \ldots, v_{\ell(C)} \rangle$, along with $f - 1$ single-vertex segments. Otherwise, the algorithm starts with $\pi(0) = 0$ and determines $\pi(t)$ for $t = 1, 2, \ldots$ iteratively in step 2, until $t = f$ or $\pi(t) = \ell(C) + 1$. In each iteration, a segment S_t of C is obtained. In step 2(a)(ii), we fix one endpoint of segment S_t to $v_{\pi(t-1)+1}$ and select the other endpoint so that the revised total edge weight of S_t is as large as possible without exceeding b_t. In the last iteration, according to step 2(a)(i), $\pi(t)$ is set equal to $\ell(C) + 1$, which is the index of the finishing endpoint of cycle C.

Example 2.1. We use the example shown in Figure 2.1 to illustrate the procedures of Algorithm 2.1, in which $f = 3$ and $\{b_1, b_2, b_3\} = \{4, 3, 17\}$. The cycle C without repeated customers is depicted in Figure 2.1(a), where $\{d_1, d_2\}$ are the depots and $\{u_1, u_2, u_3\}$ are the customers. The edge weights and vertex weights are also shown in Figure 2.1(a). For ease of exposition, we relabel the vertices as shown in Figure 2.1(b), where a cycle $C = \langle v_1, v_2, \ldots, v_7 \rangle$ is formed. Note that $\ell(C) = 6$ and $v_1 = v_7 = d_1$. Algorithm 2.1 first revises the edge weight as $w'(u, v) = w(u, v) + h(u) + h(v)$ for each edge. For instance, $w'(v_1, v_2) = 2 + 0 + 1 = 3$. The revised edge weight for each edge is depicted in Figure 2.1(b). Set $\pi(0) = 0$ and $t = 0$. Because $w'(C) = 24 > b_1$, step 2 is executed, and cycle C is split into three segments as follows: When $t = 1$, since $w'(\langle v_1, v_2 \rangle) = 3 \leq b_1$ and $w'(\langle v_1, v_2, v_3 \rangle) = 9 > b_1$, then step 2(a)(ii) sets $\pi(1) = 2$, and then step 2(b) generates $S_1 = \langle v_1, v_2 \rangle$. When $t = 2$, since $w'(\langle v_3, v_4 \rangle) = 4 > b_2$, step 2(a)(ii) sets $\pi(2) = \pi(1) + 1 = 3$, and step 2(b) generates $S_2 = \langle v_3 \rangle$. When $t = 3$, since $t = f$, then $\pi(t) = v_7$, step 2(a)(i) sets $\pi(3) = 7$, and then step 2(b) generates $S_3 = \langle v_4, v_5, v_6, v_7 \rangle$. The segments S_1, S_2 and S_3 are shown in Figure 2.1(c).

Figure 2.1: Example to illustrate Algorithm 2.1



(a) Original cycle (b) Relabeled cycle (c) Segments generated

The following lemma provides us with some important properties of the segments generated by Algorithm 2.1.

Lemma 2.2. *Given any valid input $(C, f, \{b_i\}_{1\le i\le f})$, Algorithm 2.1 runs in polynomial time and returns a set $\mathcal{S}$ of f segments $S_1, S_2, \ldots, S_f$ that satisfy the following properties:*

(i) Each vertex $v \in V(C)$ appears in at least one of the segments $S_1, S_2, \ldots, S_f$, and each customer $v \in J(C) \setminus \{v_1\}$ appears in exactly one of these segments, where v_1 is an endpoint of C.

(ii) Each of the segments S_1 and S_f has an endpoint which is the same as the endpoints of C.

(iii) $w'(S_i) \le b_i$ for all $1 \le i \le f$.

(iv) If $b_i = b$ for all $1 \le i \le f-1$, and if $f = \lceil w'(C)/b \rceil$, then $\text{cost}(\mathcal{S}) \le \max\{b, h_{\max}\}$.

Proof. Because step 2 of Algorithm 2.1 has at most f iterations and because the algorithm searches through cycle C only once, Algorithm 2.1 can be terminated in $O(f+\ell(C))$ time. The validity of properties (i), (ii) and (iii) is obvious. If $w'(C) \le b_1$,

then verifying the validity of properties (iv) and (v) is straightforward. Hence, it suffices to consider the case where $w'(C) > b$.

To prove property (iv), suppose iterations in Step 2 stop at $t = T$, where $T \leq f$. From Step 2 and Step 3, it is easy to see that $w'(S_i) \leq b_i$ for all $i \in \{1, 2, \ldots, f\} \setminus \{T\}$, and that $w'(S_T) \leq b_T$ when $T < f$. Consider the case where $T = f$. In this case, since $\pi(t)$ is non-decreasing in t, we have $w'(S_T) = w'(C) - \sum_{t=1}^{f-1}[w'(S_t) + w'(v_{\pi(t)}, v_{\pi(t)+1})]$. From Step 2(a)ii, we know $b_t < w'(S_t) + w'(v_{\pi(t)}, v_{\pi(t)+1})$ for $1 \leq t \leq f-1$. Hence, $w'(S_T) \leq w'(C) - \sum_{t=1}^{f-1} b_t = b_f = b_T$. This implies that $w'(S_T) \leq w'(C) - \sum_{t=1}^{f-1} b_t = b_f = b_T$. Thus, property (iv) is proved.

To prove property (v), suppose $b_i = b$ for all $1 \leq i \leq f-1$, and $f = \lceil w'(C)/b \rceil$, which implies $b_f \leq b$ because $\sum_{i=1}^{f} b_i = w'(C)$. For each $1 \leq i \leq f$, we have $w'(S_i) \leq b$ according to property (iv), and $w(S_i) + h(J(S_i)) \leq \max\{w'(S_i), h_{\max}\}$ by Lemma 2.1. According to property (iii), each segment in $\mathcal{S}$ has no repeated customers, which implies $h(J(S_i)) = h(S_i)$ for $1 \leq i \leq f$. Hence, $w(S_i) + h(S_i) = w(S_i) + h(J(S_i)) \leq \max\{w'(S_i), h_{\max}\} \leq \max\{b, h_{\max}\}$ for all $1 \leq i \leq f$, which implies $\text{cost}(\mathcal{S}) \leq \max\{b, h_{\max}\}$.

□

2.4 Min-max UPCP-SD

For the min-max UPCP-SD, where each of the k vehicles has an unlimited capacity and must start its path from the only depot d, we present a $\max\{3 - 2/k, 2\}$-approximation algorithm in Section 2.4.

Given an instance $\mathcal{I} = (G, \{d\}, J, w, h, k, \infty)$, our algorithm first obtains a minimum spanning tree of G and then doubles the edges of the minimum spanning tree, so as to construct a connected Euler graph H on V with each vertex having an even degree. It is well known that every connected Euler graph has a Eulerian cycle that contains every edge of the graph exactly once, and that such a Eulerian cycle can

be obtained in $O(|V|^2)$ time [52]. The algorithm short-cuts repeated vertices on the Eulerian cycle of H to obtain a cycle C that covers V, where C contains no repeated vertices except for its endpoints. Then, it applies Algorithm 2.1 to split C into a set $\mathcal{S}$ of k segments. By joining the depot d to each segment in $\mathcal{S}$, the algorithm returns a set $\mathcal{P}$ of k paths. A detailed description of this algorithm is given below:

Algorithm 2.2 (Min-max UPCP-SD)**.**

Input: Instance $\mathcal{I} = (G, \{d\}, J, w, h, k, \infty)$.

Output: A set of feasible paths $\mathcal{P} = \{P_i : 1 \leq i \leq k\}$.

1. Find a minimum spanning tree T^* of G. Double each edge of T^* to induce a connected Euler graph H on V. Obtain a Eulerian cycle on H. Short-cut repeated vertices on the Eulerian cycle to obtain a cycle C that covers V, where C contains no repeated vertices except for its endpoints. Relabel the vertices on C as $\langle v_1, v_2, \ldots, v_{\ell(C)+1} \rangle$ such that $v_1 = v_{\ell(C)+1} = d$.

2. Let $b = [w'(C) - 2w_{\max}(d)]/k$, where $w_{\max}(d) = \max_{v \in J} w(d, v)$ as defined in Section 2. Define thresholds $b_1 := b + w_{\max}(d)$, $b_i := b$ for $2 \leq i \leq k-1$, and $b_k := w'(C) - \sum_{i=1}^{k-1} b_i$. Apply Split$(C, k, \{b_i\}_{1 \leq i \leq k})$ to split C into a set of k segments, denoted $\mathcal{S} = \{S_i : 1 \leq i \leq k\}$.

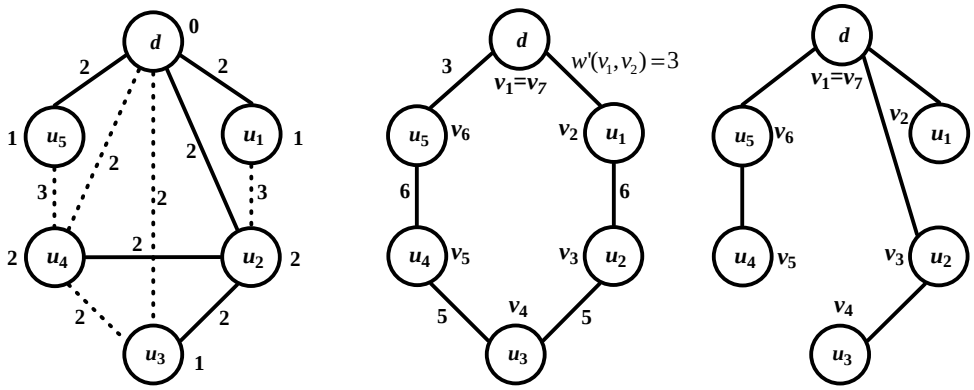
3. Set $P_1 := S_1$ and $P_k := S_k$. For $2 \leq i \leq k-1$, construct a path P_i by joining d to the closer endpoint of S_i. Return $\mathcal{P} = \{P_i : 1 \leq i \leq k\}$.

Note that in step 3, paths P_1 and P_k are set equal to S_1 and S_k, respectively. We can do so because both segments S_1 and S_k contain d as one of their endpoints.

Example 2.2. We use an example to illustrate Algorithm 2.2. In this example, $D = \{d\}$ and $k = 3$. The minimum spanning tree T^* is shown with solid lines in Figure 2.2(a). The edge weights $w(u, v)$ and vertex weights $h(v)$ are also shown in the figure. An Euler graph H is constructed by doubling the edges of T^*, and a Eulerian cycle $\langle d, u_1, d, u_2, u_3, u_2, u_4, u_2, d, u_5, d \rangle$ is obtained. A cycle $C = \langle d, u_1, u_2, u_3, u_4, u_5, d \rangle$

that covers V is obtained by short-cutting the repeated vertices, as depicted in Figure 2.2(b). Relabel the vertices from d as $v_1, v_2, \ldots, v_7$ (see Figure 2.2(b)). Figure 2.2(b) shows the revised edge weights of edges of C. Thus, $w'(C) = 28$. Since $w_{\max}(d) = 2$, we have $b = [w'(C) - 2w_{\max}(d)]/k = 8$, $b_1 = b + w_{\max}(d) = 10$, $b_2 = b = 8$, and $b_3 = w'(C) - b_1 - b_2 = 10$. Applying Split$(C, k, \{b_i\}_{1 \le i \le k})$ to split C into three segments, we obtain $S_1 = \langle v_1, v_2 \rangle$, $S_2 = \langle v_3, v_4 \rangle$, and $S_3 = \langle v_5, v_6, v_7 \rangle$. Finally, we let $P_1 = S_1$ and $P_3 = S_3$, and after joining v_1 to a closer endpoint of S_2, we obtain $P_2 = \langle v_1, v_3, v_4 \rangle$. The path set $\mathcal{P} = \{P_1, P_2, P_3\}$ is shown in Figure 2.2(c).

Figure 2.2: Example to illustrate Algorithm 2.2



(a) Min. spanning tree T^* (b) Cycle C (c) Path set $\mathcal{P}$

To analyze Algorithm 2.2, we first present a lower bound on OPT$(\mathcal{I})$.

Lemma 2.3. *For any instance $\mathcal{I} = (G, \{d\}, J, w, h, k, \infty)$ of the min-max UPCP-SD,*

$$\mathrm{OPT}(\mathcal{I}) \ge \max\left\{h_{\max},\, w_{\max}(d),\, \frac{b}{2} + \frac{w_{\max}(d)}{k}\right\},$$

where $b = [w'(C) - 2w_{\max}(d)]/k$, and C is the cycle obtained in step 1 of Algorithm 2.2.

Proof. By definition, every feasible solution to $\mathcal{I}$ must cover every customer in J, and each of its paths must start from the depot d. Thus, OPT$(\mathcal{I}) \ge h_{\max}$, and OPT$(\mathcal{I}) \ge w_{\max}(d)$. To prove OPT$(\mathcal{I}) \ge b/2 + w_{\max}(d)/k$, consider an optimal path cover $\mathcal{P}^* = \{P_i^* : 1 \le i \le k\}$. Since the paths in $\mathcal{P}^*$ are all from d and they connect

all vertices of G, we have $\sum_{i=1}^{k} w(P_i^*) \geq w(E(T^*))$ and $\sum_{i=1}^{k} h(P_i^*) \geq h(J)$, where T^* is the minimum spanning tree obtained in step 1 of Algorithm 2.2. Hence,

$$\mathrm{OPT}(\mathcal{I}) = \max_{i=1,2,\ldots,k} \{w(P_i^*)+h(P_i^*)\} \geq \frac{1}{k}\sum_{i=1}^{k} \big[w(P_i^*)+h(P_i^*)\big] \geq \frac{1}{k}\big[w(E(T^*))+h(J)\big]. \tag{2.4}$$

According to step 1 of Algorithm 2.2, we know that $J(C) = J$, and that C has at least one edge but no repeated customers. Thus, by Lemma 2.1, $w'(C) = w(C)+2h(J(C))$. Due to the triangle inequality, $w(C) \leq 2w(E(T^*))$. Therefore, from (2.4),

$$\mathrm{OPT}(\mathcal{I}) \geq \frac{1}{k}\Big[\frac{w(C)}{2} + h(J(C))\Big] = \frac{w'(C)}{2k} = \frac{b}{2} + \frac{w_{\max}(d)}{k}.$$

□

We can now establish Theorem 2.1 to show that Algorithm 2.2 achieves an approximation ratio of $\max\{3 - 2/k,\, 2\}$.

Theorem 2.1. *Given any instance $\mathcal{I}$ of the min-max UPCP-SD, Algorithm 2.2 returns a path set $\mathcal{P}$ in polynomial time, such that $\mathcal{P}$ is a feasible solution to $\mathcal{I}$ with* $\mathrm{cost}(\mathcal{P}) \leq \max\{3 - 2/k,\, 2\}\mathrm{OPT}(\mathcal{I})$.

Proof. Step 1 of Algorithm 2.2 takes $O(|V|^2)$ time. By Lemma 2.2, step 2 takes $O(k + |V|)$ time. Hence, Algorithm 2.2 has a polynomial time complexity. Clearly, every path in $\mathcal{P}$ starts from a depot. By Lemma 2.2(i), each customer in $J(V)$ appears in exactly one of the paths in $\mathcal{P}$. Hence, $\mathcal{P}$ is a feasible solution to $\mathcal{I}$.

Next, we prove that $\mathrm{cost}(\mathcal{P}) \leq \max\{3 - 2/k,\, 2\}\mathrm{OPT}(\mathcal{I})$. As defined in step 2 of Algorithm 2.2, $b_1 = b + w_{\max}(d)$ and $b_i = b$ for $2 \leq i \leq k-1$. Note that b_k is set equal to $w'(C) - \sum_{i=1}^{k-1} b_i$. It is easy to check that $b_k \leq b + w_{\max}(d)$. Hence, by Lemma 2.2(iv), we have

$$w'(S_i) \leq b + w_{\max}(d), \text{ for } i = 1 \text{ and } k, \tag{2.5}$$

$$w'(S_i) \le b, \text{ for } 2 \le i \le k-1. \tag{2.6}$$

By Lemma 2.2(ii), each of S_1 and S_k has an endpoint at the depot d. Thus, $w(P_i) + h(J(P_i)) \le w(S_i) + h(J(S_i))$ for $i = 1$ and k. For $2 \le i \le k-1$, from step 3 of Algorithm 2.2, it is easy to see that $w(P_i) + h(J(P_i)) \le w(S_i) + h(J(S_i)) + w_{\max}(d)$. Because the paths in $\mathcal{P}$ contain no repeated customers, we have $h(J(P_i)) = h(P_i)$ for $1 \le i \le k$. Thus, $w(P_i) + h(P_i) \le w(S_i) + h(J(S_i))$ for $i = 1$ and k; and $w(P_i) + h(P_i) \le w(S_i) + h(J(S_i)) + w_{\max}(d)$ for $2 \le i \le k-1$. Hence, by Lemma 2.1, we have $w(P_i) + h(P_i) \le \max\{h_{\max}, w'(S_i)\}$ for $i = 1$ and k; and $w(P_i) + h(P_i) \le \max\{h_{\max}, w'(S_i)\} + w_{\max}(d)$ for $2 \le i \le k-1$. This, together with (2.5) and (2.6), implies that $w(P_i) + h(P_i) \le \max\{h_{\max} + w_{\max}(d),\, b + w_{\max}(d)\}$ for all $1 \le i \le k$. From (2.1), we have

$$\text{cost}(\mathcal{P}) \le \max\{h_{\max} + w_{\max}(d),\, b + w_{\max}(d)\}. \tag{2.7}$$

When $k = 1$, by inequality (2.7) and Lemma 2.3, we have

$$\text{cost}(\mathcal{P}) \le \max\Big\{h_{\max} + \Big(\frac{b}{2} + w_{\max}(d)\Big),\, 2\Big(\frac{b}{2} + w_{\max}(d)\Big)\Big\} \le 2\text{OPT}(\mathcal{I}).$$

When $k \ge 2$, by inequality (2.7) and Lemma 2.3, we have

$$\begin{aligned}\text{cost}(\mathcal{P}) &\le \max\Big\{2\Big(\frac{b}{2} + \frac{w_{\max}(d)}{k}\Big) + \Big(1 - \frac{2}{k}\Big)w_{\max}(d),\, h_{\max} + w_{\max}(d)\Big\} \\ &\le \max\Big\{3 - \frac{2}{k},\, 2\Big\}\text{OPT}(\mathcal{I}).\end{aligned}$$

Combining these two cases, we obtain $\text{cost}(\mathcal{P}) \le \max\{3 - 2/k,\, 2\}\text{OPT}(\mathcal{I})$. □

Remark 2.1. As the cycle C constructed in step 1 of Algorithm 2.2 covers V and has no repeated vertices, C is an approximation solution to the traveling salesman

problem (TSP) on G. Because C is constructed by doubling the edges of a minimum spanning tree T^* of G, we get $w(C) \leq 2w(E(T^*)) \leq 2\text{OPT}(\mathcal{I})$. One may consider to change step 1 of the algorithm by applying Christofides' heuristic [19] to construct a better cycle C' through adding to T^* a minimum perfect matching of odd-degree vertices in T^*. However, since each solution to the min-max UPCP-SD is a set of paths instead of cycles, we do not have $w(C') \leq 1.5\text{OPT}(\mathcal{I})$. Thus, changing C to C' in step 1 cannot give us a significantly better approximation ratio for Algorithm 2.2.

Remark 2.2. Campbell et al. [15] showed that, for the min-max UPCP-SD, without the servicing handling time, the latest service completion time of the optimal TSP tour is no greater than $2k$ times the minimum latest service completion time for the min-max UPCP-SD. Based on this result, one can prove that Algorithm 2.2 achieves an approximation ratio of 5, because the TSP tour split by Algorithm 2.2 has a total edge weight at most twice the total edge weight of the optimal TSP tour. In this sense, we contribute to the literature of min-max path cover problem by showing a tighter approximation ratio of Algorithm 2.2, which equals to $\max\{3 - 2/k, 2\}$, for the min-max UPCP-SD.

2.5 Min-max UPCP-MD

For the min-max UPCP-MD, where each of the k vehicles has an unlimited capacity and can start its path from any depot in the depot set D, we present a 5-approximation algorithm in Section 2.5.

Given any instance $\mathcal{I} = (G, D, J, w, h, k, \infty)$ and any guess of $\text{OPT}(\mathcal{I})$, denoted by λ, the following algorithm either returns "λ is too small," which implies that $\lambda < \text{OPT}(\mathcal{I})$, or returns a feasible solution $\mathcal{P}$ to $\mathcal{I}$ with $\text{cost}(\mathcal{P}) \leq 5\lambda$. We will show that by incorporating the following algorithm into a binary search procedure, we can obtain a 5-approximation solution in polynomial time. In what follows, we use $G(J, \lambda)$ to denote a subgraph of G that contains only vertices in J and edges with

weights no greater than λ. For $1 \leq j \leq m(\lambda)$, we let $G_j(\lambda)$ denote the jth connected component of $G(J, \lambda)$, where $m(\lambda)$ is the total number of the connected components.

Algorithm 2.3 (Min-max UPCP-MD)**.**
Input: Instance $\mathcal{I} = (G, D, J, w, h, k, \infty)$; and a guess of OPT($\mathcal{I}$) denoted by $\lambda > 0$.
Output: "λ is too small," or a feasible solution $\mathcal{P}$ to $\mathcal{I}$ with cost($\mathcal{P}$) $\leq 5\lambda$.

1. If there exists $v \in J$ with $h(v) > \lambda$ or $w_{\min}(v, D) > \lambda$, then return "λ is too small."

2. For $1 \leq j \leq m(\lambda)$, find a minimum spanning tree T_j^* of each connected component $G_j(\lambda)$ of $G(J, \lambda)$. Double each edge of T_j^* to induce a connected Euler graph H_j on $V(G_j(\lambda))$. Obtain a Eulerian cycle of H_j. Short-cut repeated vertices on the Eulerian cycle to obtain a cycle C_j that covers the vertex set of $G_j(\lambda)$, that has no repeated vertices (except at the endpoints of C_j), and that contains no depot.

3. Set $k_j := \max\{\lceil w'(C_j)/(4\lambda) \rceil, 1\}$ for $1 \leq j \leq m(\lambda)$. If $\sum_{j=1}^{m(\lambda)} k_j > k$, return "$\lambda$ is too small."

4. For $1 \leq j \leq m(\lambda)$, set thresholds $b_i := 4\lambda$ for $1 \leq i \leq k_j - 1$, and $b_{k_j} := w'(C_j) - \sum_{i=1}^{k_j - 1} b_i$. Apply Split$(C_j, k_j, \{b_i\}_{1 \leq i \leq k})$ to split C_j into a set of k_j segments, denoted $\mathcal{S}_j = \{S_{j,i} : 1 \leq i \leq k_j\}$.

5. For $1 \leq j \leq m(\lambda)$ and for $1 \leq i \leq k_j$, construct a path $P_{j,i}$ by joining one endpoint of $S_{j,i}$ to its closest depot in D. If a customer on $P_{j,i}$ already exists on one of the paths that we have constructed, then modify $P_{j,i}$ by short-cutting that customer. For $1 \leq i \leq k - \sum_{j=1}^{m(\lambda)} k_j$, let $P_i := \langle d \rangle$, where d is any depot in D. Let $\mathcal{P}$ denote the set of these k paths. Return the resulting $\mathcal{P}$.

Note that unlike step 1 of Algorithm 2.2, step 2 of Algorithm 2.3 constructs a minimum spanning tree for each connected component $G_j(\lambda)$, and this connected compo-

nent does not contain any depot. In step 4 of Algorithm 2.3, Split$(C_j, k_j, \{b_i\}_{1\le i\le k})$ is applied to split each cycle into segments. Note that the endpoint of one segment may be the same as the endpoint of another segment. Thus, in step 5, when a path is formed, we short-cut those customers that are already covered by other paths. Note also that in step 5, when path $P_{j,i}$ is constructed, we may arbitrarily select one of the two endpoints of $S_{j,i}$ and then connect it to its closest depot. Next, we establish a lemma to show the correctness of Algorithm 2.3.

Lemma 2.4. *Given any instance $\mathcal{I}$ of the min-max UPCP-MD and any $\lambda > 0$, Algorithm 2.3 runs in polynomial time. If Algorithm 2.3 returns "λ is too small," then $\lambda < \mathrm{OPT}(\mathcal{I})$. Otherwise, it returns a path set $\mathcal{P}$, which is a feasible solution to $\mathcal{I}$, with $\mathrm{cost}(\mathcal{P}) \le 5\lambda$.*

Proof. Step 2 of Algorithm 2.3 takes $O(|V|^2)$ time. By Lemma 2.2, step 4 takes $O(\sum_j (k_j + |V(C_j)|)) \le O(k + |V|)$ time. Hence, Algorithm 2.3 has a polynomial time complexity.

If Algorithm 2.3 returns "λ is too small" in step 1, then it implies that $h(v) > \lambda$ or $w_{\min}(v, D) > \lambda$ for some $v \in V$. Because $h(v) \le \mathrm{OPT}(\mathcal{I})$ and $w_{\min}(v, D) \le \mathrm{OPT}(\mathcal{I})$ for any $v \in V$, we have $\lambda < \mathrm{OPT}(\mathcal{I})$. If Algorithm 2.3 returns "λ is too small" in step 3, then $\sum_{j=1}^{m(\lambda)} k_j > k$. Suppose, to the contrary, that $\mathrm{OPT}(\mathcal{I}) \le \lambda$. By the triangle inequality, there exists an optimal solution $\mathcal{P}^*$ such that each path $P \in \mathcal{P}^*$ contains no repeated customers. Because $\mathrm{cost}(\mathcal{P}^*) \le \lambda$, all customers of P must belong to the same connected component of $G(J, \lambda)$, for any $P \in \mathcal{P}^*$. For each $1 \le j \le m(\lambda)$, let $\mathcal{P}_j^*$ denote the subset of paths of $\mathcal{P}^*$ whose customers belong to $G_j(\lambda)$, and let $k_j^* = |\mathcal{P}_j^*|$. Thus, $k_j^* \ge 1$ because $G_j(\lambda)$ contains at least one customer. Since $\mathcal{P}_j^*$ for $1 \le j \le m(\lambda)$ are disjoint from each other, we have $\sum_{j=1}^{m(\lambda)} k_j^* \le |\mathcal{P}^*| = k$. By eliminating the depots from the paths in $\mathcal{P}_j^*$ and adding at most $k_j^* - 1$ edges with each edge having a weight no greater than λ, one can obtain a subgraph that spans

all vertices of $G_j(\lambda)$. This implies that

$$w(E(T_j^*)) + h(J(T_j^*)) \leq \sum_{P \in \mathcal{P}_j^*} \big[w(P) + h(P)\big] + (k_j^* - 1)\lambda \leq 2k_j^*\lambda, \tag{2.8}$$

where the latter inequality follows from the fact that each path P in $\mathcal{P}_j^*$ has a total edge and vertex weight no greater than λ. If C_j defined in step 2 of the algorithm contains at least one edge, then $w'(C_j) = w(C_j)+2h(J(C_j))$ by Lemma 2.1; otherwise, $w'(C_j) = 0$. Thus, $w'(C_j) \leq w(C_j)+2h(J(C_j))$. According to step 2 of the algorithm, $w(C_j) + 2h(J(C_j)) \leq 2[w(E(T_j^*)) + h(J(T_j^*))]$. These, together with (2.8), we have $w'(C_j) \leq 4k_j^*\lambda$. Since $k_j = \max\{\lceil w'(C_j)/(4\lambda)\rceil, 1\}$ and $k_j^* \geq 1$, we obtain $k_j \leq k_j^*$, which implies $\sum_{j=1}^{m(\lambda)} k_j \leq \sum_{j=1}^{m(\lambda)} k_j^* \leq k$, leading to a contradiction. Therefore, we conclude that $\lambda < \mathrm{OPT}(\mathcal{I})$ if the algorithm returns "λ is too small."

Next, if Algorithm 2.3 does not return "λ is too small," then it returns a path set $\mathcal{P}$ in step 5. Clearly, $|\mathcal{P}| = k$. By Lemma 2.2(i), each customer appears in exactly one of the segments in $\bigcup_{j=1}^{m(\lambda)} \mathcal{S}_j$. Thus, each customer appears exactly once in the paths of $\mathcal{P}$. Since each path in $\mathcal{P}$ starts from a depot in D, the path set $\mathcal{P}$ must be a feasible solution to $\mathcal{I}$. By Lemma 2.2(v), we have $\mathrm{cost}(\mathcal{S}_j) \leq \max\{4\lambda, h_{\max}\}$ for $1 \leq j \leq m(\lambda)$. Because "$h_{\max} \leq \lambda$" is satisfied in step 1, we obtain $\mathrm{cost}(\mathcal{S}_j) \leq 4\lambda$ for $1 \leq j \leq m(\lambda)$. Because "$w_{\min}(v, D) \leq \lambda$" is satisfied in step 1 for each $v \in J$, we conclude that $\mathrm{cost}(\mathcal{P}) \leq 5\lambda$. □

From Lemma 2.4, we can use a binary search to obtain a 5-approximation algorithm, as stated in the following theorem.

Theorem 2.2. *There exists a polynomial-time 5-approximation algorithm for the min-max UPCP-MD.*

Proof. It is easy to see that $h_{\max} \leq \mathrm{OPT}(\mathcal{I}) \leq w(E) + h(J)$. We can apply a binary search for an integer value λ^* in an interval $[h_{\max}, w(E) + h(J)]$, such that

Algorithm 2.3 will return a feasible solution to $\mathcal{I}$ when $\lambda \geq \lambda^*$, and that it will return "λ is too small" when $\lambda < \lambda^*$. Executing Algorithm 2.3 with $\lambda = \lambda^*$ yields a solution $\mathcal{P}$ with $\text{cost}(\mathcal{P}) \leq 5\lambda^* \leq 5OPT(\mathcal{I})$. The total number of iterations of this binary search is $O(\log(w(E) + h(J)))$, which is polynomial in the input size of $\mathcal{I}$. □

2.6 Summary

In this chapter, we have developed the first constant ratio approximation algorithms for two variants of a min-max uncapacitated path cover problem, where a fleet of vehicles with unlimited capacity start from the same depot or from any depot in a given set of multiple depots to serve customers located in a metric undirected graph, so as to minimize the latest service completion time. Based on the results presented in this chapter, our future work will focus on improving the worst-case approximation ratios for problems studied in this chapter, or developing constant ratio approximation algorithms for other more complicated variants of the min-max uncapacitated path cover problem.

CHAPTER 3

Approximation Algorithms for the Min-max Capacitated Path Cover Problems

3.1 Introduction

The min-max Capacitated Path Cover Problem (CPCP) is an extension of the min-max Uncapacitated Path Cover Problem (UPCP), in which each vehicle has a limited capacity Q. Unlike the min-max UPCP, when each of the k vehicles has a limited capacity of Q, the min-max CPCP allows at most Q customers to be served by each trip of the vehicle. In other words, the vehicles need to replenish at a depot in the depot set before servicing at most Q customers further. As far as we know, there are no constant ratio approximation algorithms available in the literature for the min-max CPCP. Moreover, due to the capacity restriction, the cycle splitting procedure designed for the min-max UPCP can not be applied directly to the min-max CPCP.

In this chapter, we develop constant ratio approximation algorithms for two variants of the min-max CPCP in a metric undirected network. The remainder of this chapter is organized as follows. Section 3.2 introduces the definition of the min-max CPCP, and the notation that are used throughout this chapter. In Section 3.3, we develop a cycle splitting procedure by customer numbers, which has been extensively applied in the development of approximation algorithms in this chapter. For the

min-max CPCP with a single depot available for all the vehicles, we develop the first constant ratio approximation algorithm in Section 3.4, which achieves an approximation ratio of $\max\{5-2/k, 4\}$. For the min-max CPCP with multiple depots available for all the vehicles, we develop the first constant ratio approximation algorithm in Section 3.5, which achieves an approximation ratio of 7. We then show possible extensions of our techniques in Section 3.6 to develop approximation algorithms for other capacitated vehicle routing problems. Finally, we summarize this chapter in Section 3.7.

3.2 Notation and Problem Definition

The min-max CPCP is an extension of the min-max UPCP, where each of the k identical vehicles has a capacity Q and needs to replenish at some depot in the depot set before servicing at most Q customers further. Therefore, we can denote an instance of the min-max capacitated path cover problem as (G, D, J, w, h, k, Q). Moreover, similar to the min-max UPCP, depending on whether D contains only one depot or multiple depots, the problem has the following two typical variants: the min-max capacitated path cover problem with single depot (CPCP-SD), and the min-max capacitated path cover problem with multiple depots (CPCP-MD).

Recall that, in Chapter 2, we have defined a *segment* of path P as a contiguous subsequence of its vertices (which is allowed to be empty). Thus, we can define a *trip* of P as a segment of P consisting of at least two vertices such that the segment starts from either the starting endpoint of P or a depot, ends at either the finishing endpoint of P or a depot, and contains no depot in between. In other words, a trip of P is a segment $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ of P for any $1 \leq i < j \leq \ell(P) + 1$ such that either $i = 1$ or $v_i \in D$, that either $j = \ell(P) + 1$ or $v_j \in D$, and that vertices $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$ are elements of J. Hence, if P has at least two vertices, the total number of trips in P equals one plus the total number of occurrences of depots on P (repeated depots are

counted repeatedly), excluding the endpoints. In particular, if P has only one vertex, the total number of trips in P equals one, and if P has no vertex, the total number of trips in P equals zero.

Accordingly, the min-max CPCP can be formulated as follows:

Min-max Capacitated Path Cover Problem

Instance: $\mathcal{I} = (G, D, J, w, h, k, Q)$, where $G = (V, E)$ is a complete undirected graph; $D \subset V$; $J = V \setminus D$; w is a metric function on E; h is a function on V with $h(d) = 0$ for each $d \in D$; and k, Q are positive integers.

Feasible solution: An exact path cover $\mathcal{P}$ of $\mathcal{I}$ that consists of k paths, such that each path in $\mathcal{P}$ contains at most Q customers in each of its trips.

Objective: $\min_{\mathcal{P}} cost(\mathcal{P})$.

Similar to the min-max UPCP, we assume in this chapter that all edge and vertex weights are integers.

3.3 Cycle Splitting by Customer Numbers

Besides the revised weight $w'(u, v) = w(u, v) + h(u) + h(v)$ for each edge $(u, v) \in E$ and the cycle splitting procedure by revised edge weight defined in Chapter 2, approximation algorithms developed in this chapter also rely on a new cycle splitting procedure. This cycle splitting procedure not only splits a cycle into a set of segments but also joins these segments together by inserting depots between segments, so that a new cycle is constituted with every trip having no more than Q customer.

This cycle splitting procedure is denoted by $\text{Split}(C', p)$. Given a cycle $C' = \langle v'_1, v'_2, \ldots, v'_{\ell(C')+1} \rangle$ which contains exactly one depot located at $v'_1 = v'_{\ell(C')+1}$ and has at least one customer but no repeated customers, and given a vertex index p, where $1 \leq p \leq Q'$ and $Q' = \min\{Q, |J(C')|\}$, this procedure first splits cycle C' into

$\lceil (|J(C')| - p)/Q' \rceil + 1$ segments, with the first segment containing the first p customers on C', and with each of the other segments containing at most Q' customers. It then joins these segments together by inserting depots between segments, so that a new cycle C_p is constituted, and that C_p has its endpoints at a depot, covers all vertices of C', and has no trip containing more than Q customers. A description of this cycle splitting procedure is given below:

Algorithm 3.1 (Split(C', p))**.**

Input: A cycle $C' = \langle v'_1, v'_2, \ldots, v'_{\ell(C')+1} \rangle$ which contains exactly one depot located at $v'_1 = v'_{\ell(C')+1} \in D$, contains at least one customer, but contains no repeated customers; and a vertex index p, where $1 \leq p \leq Q'$ and $Q' = \min\{Q, |J(C')|\}$.

Output: A cycle C_p that covers all customers of C' with both of its endpoints at v'_1, with the first trip containing p customers, and with no trip containing more than Q customers.

1. Let $n_p = \lceil (|J(C')| - p)/Q' \rceil$. If $n_p = 0$, then return C' as C_p. Otherwise, go to step 2.

2. Split cycle C' into a set of $(n_p + 1)$ segments $\mathcal{S}_p = \{S_{p,i} : 0 \leq i \leq n_p\}$ as follows:

   (a) Set segment $S_{p,0} := \langle v'_1, \ldots, v'_{p+1} \rangle$.

   (b) For $1 \leq i \leq n_p - 1$, set segment $S_{p,i} := \langle v'_{1+p+(i-1)Q'+1}, \ldots, v'_{1+p+iQ'} \rangle$.

   (c) Set segment $S_{p,n_p} := \langle v'_{1+p+(n_p-1)Q'+1}, \ldots, v'_{|\ell(C')|+1} \rangle$.

3. Construct cycle C_p from the segments in $\mathcal{S}_p$ as follows; For each $1 \leq i \leq n_p$, insert two edges $(d_{p,i}, v'_{1+p+(i-1)Q'})$ and $(d_{p,i}, v'_{1+p+(i-1)Q'+1})$ to connect segments $S_{p,i-1}$ and $S_{p,i}$, where $d_{p,i} \in D$ is the depot closest to $v'_{1+p+(i-1)Q'}$.

4. Relabel the vertices on C_p such that both of its endpoints are at depot v'_1. Return C_p.

If $n_p = 0$, which implies that $J(C') = p$, then Algorithm 3.1 terminates in step 1 and returns C' as its output C_p. Otherwise, the algorithm splits C' in step 2 into a set of $(n_p + 1)$ segments. Note that in step 2(a), segment $S_{p,0}$ consists of the depot v'_1 and the first p customers on C'. In step 2(b), segment $S_{p,i}$ contains exactly Q' customers for $1 \leq i \leq n_p - 1$. In step 2(c), segment S_{p,n_p} contains the remaining vertices on C', which include the depot $v'_{|\ell(C')|+1}$ and at most Q' customers (because $|J(C')| = \ell(C') - 1$ and $v'_1 = v'_{\ell(C')+1} \in D$). Thus, a cycle C_p is generated in step 3, because $S_{p,0}$ and S_{p,n_p} share a common endpoint at $v'_1 = v'_{\ell(C')+1}$.

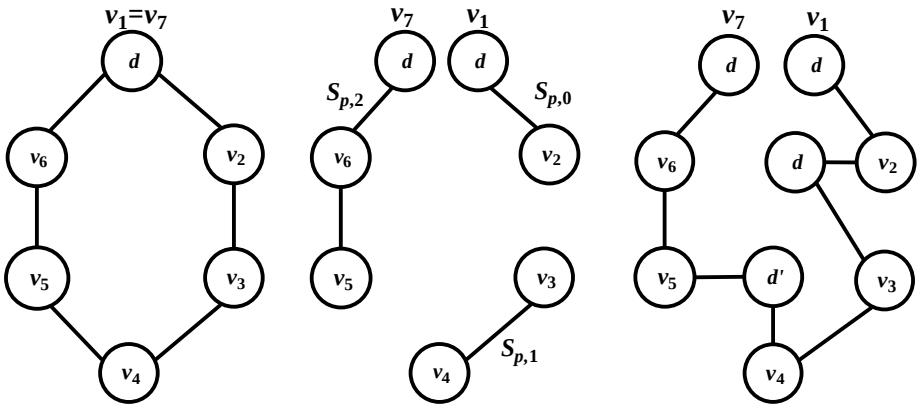
Example 3.1. We use an example to illustrate the Split(C', p) procedure. In this example, $p = 1$, $Q = 2$, and $D = \{d, d'\}$. The cycle $C' = \langle v_1, v_2, v_3, v_4, v_5, v_6, v_7 \rangle$ is given in Figure 3.1(a), where $v_1 = v_7 = d$ and v_2, v_3, v_4, v_5, v_6 are customers. Customers v_2 is closer to d than to d', while customers v_3, v_4, v_5, v_6 are closer to d' than to d. Note that $Q' = \min\{Q, J(C')\} = 2$ and $n_p = \lceil (|J(C')| - p)/Q' \rceil = 2$. Step 2 of the algorithm constructs segments $S_{p,0} = \langle d, v_2 \rangle$, $S_{p,1} = \langle v_3, v_4 \rangle$, and $S_{p,2} = \langle v_5, v_6, d \rangle$ as depicted in Figure 3.1(b). In step 3(b), edges (d, v_2) and (d, v_3) are inserted to connect $S_{p,0}$ with $S_{p,1}$, while edges (d', v_4) and (d', v_5) are inserted to connect $S_{p,1}$ with $S_{p,2}$. Hence, a cycle $C_p = \langle d, v_2, d, v_3, v_4, d', v_5, v_6, d \rangle$ is obtained as shown in Figure 3.1(c).

Lemma 3.1. *Given any valid input (C', p) with $1 \leq p \leq Q'$ and $Q' = \min\{Q, |J(C')|\}$, Algorithm 3.1 runs in $O(\ell(C'))$ time and returns a cycle C_p, such that C_p has its endpoints at a depot, covers $V(C')$, contains no repeated customers, and that each trip of C_p contains at most Q customers. Furthermore,*

$$\min_{1 \leq p \leq Q'} w(C_p) \leq w(C') + \frac{2}{Q'} \sum_{v \in J(C')} w_{\min}(v, D). \tag{3.1}$$

Proof. It is easy to verify that Algorithm 3.1 runs in $O(\ell(C'))$ time, that C_p has its endpoints at a depot, covers $V(C')$, and contains no repeated customers, and

Figure 3.1: Example to illustrate Algorithm 3.1



(a) Given cycle C' (b) Set of segments $\mathcal{S}_p$ (c) Cycle C_p

that each trip of C_p contains at most Q customers. If $n_p = 0$, then verifying the validity of (3.1) is straightforward. Hence, it is sufficient to prove inequality (3.1) for the case where $n_p \geq 1$. Consider segments $S_{p,0}, S_{p,1}, \ldots, S_{p,n_p}$ obtained in step 2 of Algorithm 3.1. Clearly, each $S_{p,i}$ contains no repeated vertices, and

$$w(S_{p,0}) + \sum_{i=1}^{n_p} \Big[w(v'_{1+p+(i-1)Q'}, v'_{1+p+(i-1)Q'+1}) + w(S_{p,i}) \Big] \leq w(C').$$

Since $d_{p,i}$ is the closest to $v'_{1+p+(i-1)Q'}$ among those depots in D, by this fact,we have $w(d_{p,i}, v'_{1+p+(i-1)Q'}) = w_{\min}(v'_{1+p+(i-1)Q'}, D)$. By the triangle inequality, we have

$$\begin{aligned}
&w(d_{p,i}, v'_{1+p+(i-1)Q'}) + w(d_{p,i}, v'_{1+p+(i-1)Q'+1}) \\
&\quad \leq w(d_{p,i}, v'_{1+p+(i-1)Q'}) + \Big[w(d_{p,i}, v'_{1+p+(i-1)Q'}) + w(v'_{1+p+(i-1)Q'}, v'_{1+p+(i-1)Q'+1}) \Big] \\
&\quad \leq 2w_{\min}(v'_{1+p+(i-1)Q'}, D) + w(v'_{1+p+(i-1)Q'}, v'_{1+p+(i-1)Q'+1}).
\end{aligned}$$

Hence,

$$\begin{aligned} w(C_p) &= w(S_{p,0}) + \sum_{i=1}^{n_p} \Big[w(d_{p,i}, v'_{1+p+(i-1)Q'}) + w(d_{p,i}, v'_{1+p+(i-1)Q'+1}) + w(S_{p,i}) \Big] \\ &\leq w(S_{p,0}) + \sum_{i=1}^{n_p} \Big[w(v'_{1+p+(i-1)Q'}, v'_{1+p+(i-1)Q'+1}) + w(S_{p,i}) \Big] \\ &+ \sum_{i=1}^{n_p} 2w_{\min}(v'_{1+p+(i-1)Q'}, D) \\ &\leq w(C') + 2\sum_{i=1}^{n_p} w_{\min}(v'_{1+p+(i-1)Q'}, D), \end{aligned}$$

which implies that

$$\min_{1\leq p\leq Q'} w(C_p) \leq \frac{1}{Q'}\sum_{p=1}^{Q'} w(C_p) \leq w(C') + \frac{2}{Q'}\sum_{v\in J(C')} w_{\min}(v, D).$$

□

3.4 Min-max CPCP-SD

For the min-max CPCP-SD, where each of the k vehicles has a capacity Q and must start its path from the only depot d, we present a $\max\{5-2/k, 4\}$-approximation algorithm in Section 3.4.

The approximation algorithm for the min-max CPCP-SD is similar to Algorithm 2.2. The main difference is that after obtaining an initial cycle, it applies Algorithm 3.1 (i.e., the Split(C', p) procedure) to construct a new cycle with each trip containing at most Q customers. When Algorithm 3.1 is applied, the parameter p is selected in such a way that the total edge weight in the cycle is minimized. A detailed description of this algorithm is given below:

Algorithm 3.2 (Min-max CPCP-SD)**.**

Input: Instance $\mathcal{I} = (G, \{d\}, J, w, h, k, Q)$.

Output: A feasible solution to $\mathcal{I}$ denoted by $\mathcal{P} = \{P_i : 1 \le i \le k\}$.

1. (Same as step 1 of Algorithm 2.2.) Find a minimum spanning tree T^* of G. Double each edge of T^* to induce a connected Euler graph H on V. Obtain an Eulerian cycle on H. Short-cut repeated vertices on the Eulerian cycle to obtain a cycle C' that covers V, where C' contains no repeated vertices except for its endpoints. Relabel the vertices on C' as $\langle v'_1, v'_2, \ldots, v'_{\ell(C')+1}\rangle$ such that $v'_1 = v'_{\ell(C')+1} = d$.

2. Let $Q' = \min\{Q, |J(C')|\}$. For each $1 \le p \le Q'$, apply Split(C', p) to obtain a new cycle C_p that covers $V(C')$. Among all these cycles, let C be the one with the smallest total edge weight (ties broken arbitrarily). Relabel the vertices on C as $\langle v_1, v_2, \ldots, v_{\ell(C)+1}\rangle$ such that $v_1 = v_{\ell(C)+1} = d$.

3. (Same as step 2 of Algorithm 2.2.) Let $b = [w'(C) - 2w_{\max}(d)]/k$. Define thresholds $b_1 := b + w_{\max}(d)$, $b_i := b$ for $2 \le i \le k-1$, and $b_k := w'(C) - \sum_{i=1}^{k-1} b_i$. Apply Split$(C, k, \{b_i\}_{1\le i\le k})$ to split C into a set of k segments, denoted $\mathcal{S} = \{S_i : 1 \le i \le k\}$.

4. (Same as step 3 of Algorithm 2.2.) Set $P_1 := S_1$ and $P_k := S_k$. For $2 \le i \le k-1$, construct a path P_i by joining d to the closer endpoint of S_i. Return $\mathcal{P} = \{P_i : 1 \le i \le k\}$.

To analyze Algorithm 3.2, we first present a lower bound on OPT$(\mathcal{I})$.

Lemma 3.2. *For any instance $\mathcal{I} = (G, \{d\}, J, w, h, k, Q)$ of the min-max CPCP-SD,*

$$\mathrm{OPT}(\mathcal{I}) \ge \max\left\{h_{\max},\, w_{\max}(d),\, \frac{w'(C)}{4k}\right\},$$

where C is the cycle obtained in step 2 of Algorithm 3.2.

Proof. Clearly, $\mathrm{OPT}(\mathcal{I}) \ge h_{\max}$ and $\mathrm{OPT}(\mathcal{I}) \ge w_{\max}(d)$. To prove $\mathrm{OPT}(\mathcal{I}) \ge w'(C)/(4k)$, consider an optimal path cover $\mathcal{P}^* = \{P_i^* : 1 \le i \le k\}$. For each

$1 \leq i \leq k$, we assume that each trip on P_i^* contains at least one customer (clearly, there exists an optimal solution to the min-max CPCP-SD in which every trip on a path contains at least one customer). Let $\eta(i)$ denote the total number of times that P_i^* leaves depot d. Thus, P_i^* contains exactly $\eta(i)$ trips, denoted $S_{i,1}^*, S_{i,2}^*, \ldots, S_{i,\eta(i)}^*$. For $1 \leq j \leq \eta(i)$, let $w_{i,j}^* = \max_{v \in J(S_{i,j}^*)} w(d,v)$. Since $S_{i,j}^*$ contains depot d, by the triangle inequality, we have $w_{i,j}^* \leq w(S_{i,j}^*)$. Because $|J(S_{i,j}^*)| \leq Q'$ and $\sum_{j=1}^{\eta(i)} w(S_{i,j}^*) = w(P_i^*)$, we have

$$\sum_{v \in J(P_i^*)} \frac{w(d,v)}{Q'} = \sum_{j=1}^{\eta(i)} \sum_{v \in J(S_{i,j}^*)} \frac{w(d,v)}{Q'} \leq \sum_{j=1}^{\eta(i)} w_{i,j}^* \leq \sum_{j=1}^{\eta(i)} w(S_{i,j}^*) = w(P_i^*),$$

which implies that

$$\sum_{v \in J} \frac{w(d,v)}{Q'} \leq \sum_{i=1}^{k} \sum_{v \in J(P_i^*)} \frac{w(d,v)}{Q'} \leq \sum_{i=1}^{k} w(P_i^*) \leq k \cdot \mathrm{OPT}(\mathcal{I}). \tag{3.2}$$

Let C' be the cycle obtained in step 1 of Algorithm 3.2. Note that $J(C') = J$. Since $w_{\min}(v, D) = w(d,v)$ for $v \in J(C')$, by Lemma 3.1, we have

$$w(C) = \min_{1 \leq p \leq Q'} w(C_p) \leq w(C') + 2 \sum_{v \in J(C')} \frac{w(d,v)}{Q'} = w(C') + 2 \sum_{v \in J} \frac{w(d,v)}{Q'}. \tag{3.3}$$

Note that paths in $\mathcal{P}^*$ span all vertices of G. Thus, $w(E(T^*)) \leq \sum_{i=1}^{k} w(P_i^*)$. Because $h(J) \leq \sum_{i=1}^{k} h(J(P_i^*))$ and $w(P_i^*) + h(J(P_i^*)) \leq w(P_i^*) + h(P_i^*) \leq \mathrm{OPT}(\mathcal{I})$ for $1 \leq i \leq k$, we have

$$w(E(T^*)) + h(J) \leq k \cdot \mathrm{OPT}(\mathcal{I}). \tag{3.4}$$

Since C covers V, we have $J(C) = J$. By Lemma 2.1, we get that

$$w'(C) = w(C) + 2h(J(C)) = w(C) + 2h(J). \tag{3.5}$$

From step 1 of Algorithm 3.2, and by the triangle inequality, we have

$$w(C') \leq 2w(E(T^*)). \tag{3.6}$$

Combining (3.2)–(3.6), we obtain

$$\begin{aligned} \frac{w'(C)}{4k} &= \frac{w(C)+2h(J)}{4k} \leq \frac{w(C')}{4k} + \frac{1}{2k}\sum_{v\in J}\frac{w(d,v)}{Q'} + \frac{h(J)}{2k} \\ &\leq \frac{w(C')}{4k} + \frac{\mathrm{OPT}(\mathcal{I})}{2} + \frac{h(J)}{2k} \leq \frac{w(E(T^*))+h(J)}{2k} + \frac{\mathrm{OPT}(\mathcal{I})}{2} \leq \mathrm{OPT}(\mathcal{I}). \end{aligned}$$

□

We can now establish Theorem 3.1 to show that Algorithm 3.2 achieves an approximation ratio of $\max\{5-2/k,\, 4\}$.

Theorem 3.1. *Given any instance $\mathcal{I}$ of the min-max CPCP-SD, Algorithm 3.2 returns a path set $\mathcal{P}$ in polynomial time, such that $\mathcal{P}$ is a feasible solution to $\mathcal{I}$ with* $\mathrm{cost}(\mathcal{P}) \leq \max\{5-2/k,\, 4\}\mathrm{OPT}(\mathcal{I})$.

Proof. Steps 1 and 2 of Algorithm 3.2 take $O(|V|^2)$ time. By Lemma 2.2, step 4 takes $O(k+|V|)$. Hence, the algorithm has a polynomial time complexity. To see that the path set $\mathcal{P}$ returned by Algorithm 3.2 is a feasible solution to $\mathcal{I}$, consider the cycle C constructed in step 2. Since C covers V, by Lemma 2.2(i), each customer in J must appear in at least one of the segments obtained in step 3. This implies that $\mathcal{P}$ is a path cover of $\mathcal{I}$. Moreover, since every trip of C contains at most Q customers, every segment of C in $\mathcal{S}$ must also be so. Thus, each trip of the paths in $\mathcal{P}$ generated by step 4 must contain at most Q customers. Therefore, $\mathcal{P}$ is a feasible solution to $\mathcal{I}$.

Next, we prove that $\mathrm{cost}(\mathcal{P}) \leq \max\{5-2/k,\, 4\}\mathrm{OPT}(\mathcal{I})$. Notice that the cycle C obtained in step 2 has both of its endpoints at depot d. As $\mathcal{P}$ is constructed from C by steps 3 and 4 of Algorithm 3.2, which are the same as steps 2 and 3 of Algorithm 2.2,

by following the same proof of Theorem 2.1, it is easy to check that the inequality (2.7) also holds for the path set $\mathcal{P}$ generated by Algorithm 3.2. Thus,

$$\text{cost}(\mathcal{P}) \leq \max\{b + w_{\max}(d),\, h_{\max} + w_{\max}(d)\}, \tag{3.7}$$

where $b = [w'(C) - 2w_{\max}(d)]/k$. When $k = 1$, by inequality (3.7) and Lemma 3.2, we have

$$\text{cost}(\mathcal{P}) \leq \max\{w'(C),\, h_{\max} + w_{\max}(d)\} \leq 4\text{OPT}(\mathcal{I}).$$

When $k \geq 2$, by inequality (3.7) and Lemma 3.2, we have

$$\text{cost}(\mathcal{P}) \leq \max\left\{4\frac{w'(C)}{4k} + \left(1 - \frac{2}{k}\right)w_{\max}(d),\, h_{\max} + w_{\max}(d)\right\} \leq \max\left\{5 - \frac{2}{k},\, 2\right\}\text{OPT}(\mathcal{I}).$$

Combining these two cases, we obtain $\text{cost}(\mathcal{P}) \leq \max\{5 - 2/k,\, 4\}\text{OPT}(\mathcal{I})$. □

3.5 Min-max CPCP-MD

For the min-max CPCP-MD, where each of the k vehicles has a capacity Q and can start its path from any depot in D, we present a 7-approximation algorithm in Section 3.5.

Our approximation algorithm for the min-max CPCP-MD is similar to Algorithm 2.3. Given any instance $\mathcal{I} = (G, D, J, w, h, k, Q)$ and any guess of $\text{OPT}(\mathcal{I})$, denoted by λ, the following algorithm either returns "λ is too small," which implies that $\lambda < \text{OPT}(\mathcal{I})$, or returns a feasible solution $\mathcal{P}$ to $\mathcal{I}$ with $\text{cost}(\mathcal{P}) \leq 7\lambda$. We will show that by incorporating the following algorithm into a binary search procedure, we can obtain a 7-approximation solution in polynomial time. We use $G(J, \lambda)$ to denote a subgraph of G that contains only vertices in J and edges with weights no greater than λ. For $1 \leq j \leq m(\lambda)$, we let $G_j(\lambda)$ denote the jth connected component of $G(J, \lambda)$, where $m(\lambda)$ denotes the total number of the connected components.

Algorithm 3.3 (Min-max CPCP-MD)**.**

Input: Instance $\mathcal{I} = (G, D, J, w, h, k, Q)$; and a guess of $\mathrm{OPT}(\mathcal{I})$ denoted by $\lambda > 0$.

Output: "λ is too small," or a feasible solution $\mathcal{P}$ to $\mathcal{I}$ with $\mathrm{cost}(\mathcal{P}) \leq 7\lambda$.

1. (Same as step 1 of Algorithm 2.3.) If there exists $v \in J$ with $h(v) > \lambda$ or $w_{\min}(v, D) > \lambda$, then return "λ is too small."

2. (Similar to step 2 of Algorithm 2.3.) For $1 \leq j \leq m(\lambda)$, do the following steps for each connected component $G_j(\lambda)$ of $G(J, \lambda)$:

   (a) Find a minimum spanning tree T_j^* of $G_j(\lambda)$. Double each edge of T_j^* to induce a connected Euler graph H_j on $V(G_j(\lambda))$. Obtain a Eulerian cycle of H_j. Short-cut repeated vertices on the Eulerian cycle to obtain a cycle C_j'' that covers the vertex set of $G_j(\lambda)$, that has no repeated vertices, and that contains no depot.

   (b) Let v_j'' be any vertex on C_j''. Let d_j'' be the depot which is the closest to v_j''. Insert d_j'' into C_j'' immediately next to v_j'' to form a cycle C_j', whose vertices are relabeled as $\langle v'_{j,1}, v'_{j,2} \ldots, v'_{j,\ell(C_j')+1} \rangle$ such that $v'_{j,1} = v'_{j,\ell(C_j')+1} = d_j'' \in D$.

3. Repeat the following steps for $j = 1, 2, \ldots, m(\lambda)$:

   (a) For each p with $1 \leq p \leq Q_j'$, where $Q_j' = \min\{Q, |J(C_j')|\}$, apply $\mathrm{Split}(C_j', p)$ to obtain a new cycle $C_{j,p}$, which covers $V(C_j')$, has both its endpoints at a depot in D, and has no trip containing more than Q customers.

   (b) Among all these cycles, let C_j be the one with the minimum total edge weight.

4. (Similar to step 3 of Algorithm 2.3.) Set $k_j := \max\{\lceil (w'(C_j) - 2\lambda)/(6\lambda) \rceil, 1\}$ for $1 \leq j \leq m(\lambda)$. If $\sum_{j=1}^{m(\lambda)} k_j > k$, return "$\lambda$ is too small."

5. (Similar to step 4 of Algorithm 2.3.) For $1 \leq j \leq m(\lambda)$, set thresholds $b_1 := 7\lambda$, $b_i := 6\lambda$ for $2 \leq i \leq k_j - 1$, and $b_{k_j} := w'(C_j) - \sum_{i=1}^{k_j-1} b_i$. Apply $\text{Split}(C, k, \{b_i\}_{1 \leq i \leq k})$ to split each C_j into a set of k_j segments, denoted $\mathcal{S}_j = \{S_{j,i} : 1 \leq i \leq k_j\}$.

6. (Similar to Step 5 of Algorithm 2.3): For $1 \leq j \leq m(\lambda)$, set paths $P_{j,1} := S_{j,1}$ and $P_{j,k_j} := S_{j,k_j}$. For $2 \leq i \leq k_j - 1$, construct a path $P_{j,i}$ by joining one endpoint of $S_{j,i}$ to its closest depot in D. If a customer on $P_{j,i}$ already exists on one of the paths that we have constructed, then modify $P_{j,i}$ by short-cutting that customer. For $1 \leq i \leq k - \sum_{j=1}^{m(\lambda)} k_j$, let $P_i := \langle d \rangle$, where d is any depot in D. Let $\mathcal{P}$ denote the set of these k paths.

Next, we establish a lemma to show the correctness of Algorithm 3.3.

Lemma 3.3. *Given any instance $\mathcal{I}$ of the min-max CPCP-MD and any $\lambda > 0$, Algorithm 3.3 runs in polynomial time. If Algorithm 3.3 returns "λ is too small," then $\lambda < \text{OPT}(\mathcal{I})$. Otherwise, it returns a path set $\mathcal{P}$, which is a feasible solution to $\mathcal{I}$, with $\text{cost}(\mathcal{P}) \leq 7\lambda$.*

Proof. It is easy to see that Algorithm 3.3 runs in polynomial time. If Algorithm 3.3 returns "λ is too small" in step 1, then as shown in the proof of Lemma 2.4, we have $\lambda < \text{OPT}(\mathcal{I})$. We now consider the case where Algorithm 3.3 returns "λ is too small" in step 4. In this case,

$$\sum_{j=1}^{m(\lambda)} k_j > k, \tag{3.8}$$

where $k_j = \max\{\lceil (w'(C_j) - 2\lambda)/(6\lambda) \rceil, 1\}$ as defined in step 4. Suppose, to the contrary, that $\lambda \geq \text{OPT}(\mathcal{I})$. By the triangle inequality, there exists an optimal solution $\mathcal{P}^*$ such that each path $P \in \mathcal{P}^*$ contains no repeated customers. Because $\text{cost}(\mathcal{P}^*) \leq \lambda$, all customers of P must belong to the same connected component of $G(J, \lambda)$, for any $P \in \mathcal{P}^*$. For each $1 \leq j \leq m(\lambda)$, let $\mathcal{P}_j^*$ denote the subset of paths

of $\mathcal{P}^*$ whose customers belong to $G_j(\lambda)$, and let $k_j^* = |\mathcal{P}_j^*|$. Because $G_j(\lambda)$ contains at least one customer, we obtain $k_j^* \geq 1$. Since $\mathcal{P}_j^*$ for $1 \leq j \leq m(\lambda)$ are disjoint from each other, we have $\sum_{j=1}^{m(\lambda)} k_j^* \leq |\mathcal{P}^*| = k$. For each $1 \leq j \leq m(\lambda)$, by following the same argument as in the derivation of (3.2), we obtain

$$\sum_{v \in J(G_j(\lambda))} \frac{w_{\min}(v, D)}{Q'_j} \leq k_j^* \cdot \mathrm{OPT}(\mathcal{I}). \tag{3.9}$$

Also, by following the same argument as in the derivation of (2.8), we obtain

$$w(E(T_j^*)) + h(J(T_j^*)) \leq 2k_j^* \lambda. \tag{3.10}$$

By Lemma 3.1, $J(C_j) = J(C'_j) = J(G_j(\lambda))$. Furthermore, $w(C_j) \leq w(C'_j) + 2\sum_{v \in J(C_j)} w_{\min}(v, D)/Q'_j$. This, together with (3.9), implies that

$$w(C_j) \leq w(C'_j) + 2k_j^* \cdot \mathrm{OPT}(\mathcal{I}). \tag{3.11}$$

According to step 2(b) and by the triangle inequality, we have $w(C'_j) \leq w(C''_j) + 2w_{\min}(v''_j, D)$, where v''_j is a vertex on C'_j defined in step 2(b). Because $w(C''_j) \leq 2w(E(T_j^*))$ and $w_{\min}(v''_j, D) \leq \mathrm{OPT}(\mathcal{I})$, we have $w(C'_j) \leq 2w(E(T_j^*)) + 2\mathrm{OPT}(\mathcal{I})$. Note that $J(C_j) = J(T_j^*)$. Hence,

$$w(C'_j) + 2h(J(C_j)) \leq 2w(E(T_j^*)) + 2h(J(T_j^*)) + 2\mathrm{OPT}(\mathcal{I}). \tag{3.12}$$

Combining (3.10), (3.11), and (3.12), we obtain $w(C_j) + 2h(J(C_j)) \leq 6k_j^*\lambda + 2\mathrm{OPT}(\mathcal{I})$, implying that

$$w(C_j) + 2h(J(C_j)) \leq (6k_j^* + 2)\lambda.$$

If C_j defined in step 3(b) of the algorithm contains at least one edge, then $w'(C_j) = w(C_j) + 2h(J(C_j))$ by Lemma 2.1; otherwise, $w'(C_j) = 0$. Thus, $w'(C_j) \leq w(C_j) +$

$2h(J(C_j)) \leq (6k_j^* + 2)\lambda$, which implies $k_j^* \geq \max\{\lceil (w'(C_j) - 2\lambda)/(6\lambda)\rceil, 1\} = k_j$ because $k_j \geq 1$. Thus, $\sum_{j=1}^{m(\lambda)} k_j \leq \sum_{j=1}^{m(\lambda)} k_j^* \leq k$, which contradicts (3.8). Therefore, we conclude that $\lambda < \mathrm{OPT}(\mathcal{I})$ if the algorithm returns "λ is too small."

Next, we consider the case where Algorithm 3.3 does not return "λ is too small." In this case, the algorithm returns a path set $\mathcal{P}$ in step 6. Similar to the proof of Theorem 3.1, it can be verified that $\mathcal{P}$ is a feasible solution to $\mathcal{I}$. To prove $\mathrm{cost}(\mathcal{P}) \leq 7\lambda$, consider each connected component $G_j(\lambda)$ of $G(J, \lambda)$, where $1 \leq j \leq m(\lambda)$. Notice that step 5 of Algorithm 3.3 applies Algorithm 2.1 to split C_j into a set $\mathcal{S}_j$ of at most k_j segments with $b_1 = 7\lambda$ and $b_i = 6\lambda$ for $1 \leq i \leq k_j - 1$. Also, it is easy to check that $b_{k_j} \leq 7\lambda$. Thus, by Lemma 2.1 and Lemma 2.2(iv), $w(S_{j,i}) + h(J(S_{j,i})) \leq \max\{h_{\max}, w'(S_{j,i})\} \leq \max\{h_{\max}, b_i\}$ for $i = 1, 2, \ldots, k_j$. Moreover, by step 1, $h_{\max} \leq \lambda$. Hence, $w(S_{j,1}) + h(J(S_{j,1})) \leq 7\lambda$, $w(S_{j,k_j}) + h(J(S_{j,k_j})) \leq 7\lambda$, and $w(S_{j,i}) + h(J(S_{j,i})) \leq 6\lambda$ for $2 \leq i \leq k_j - 1$. From step 6, we know that $P_{j,1} = S_{j,1}$ and $P_{j,k_j} = S_{j,k_j}$. For $2 \leq i \leq k_j - 1$, $P_{j,i}$ is formed by joining one endpoint of $S_{j,i}$ to its closest depot in D, which implies that $w(P_{j,i}) \leq w(S_{j,i}) + \lambda$ (since, according to step 1, $w_{\min}(v, D) \leq \lambda$ for all $v \in J$). Hence, $w(P_{j,i}) + h(P_{j,i}) \leq 7\lambda$ for all $1 \leq i \leq k_j$. Therefore, $\mathrm{cost}(\mathcal{P}) \leq 7\lambda$. □

From Lemma 3.3, we can use a binary search to obtain an approximation algorithm as stated in the following theorem. The proof of this theorem follows the same argument as that of Theorem 2.2 and is omitted.

Theorem 3.2. *There exists a 7-approximation algorithm for the min-max CPCP-MD.*

3.6 Extensions

The techniques developed for the min-max capacitated path cover problems can be extended to other min-max capacitated vehicle routing problems. In this section,

we demonstrate the possible extensions to two such problems, namely, the min-max capacitated cycle cover problem with a single depot (CCCP-SD) and the min-max CPCP-SD with single trips (CPCP-SD-ST).

3.6.1 Min-max CCCP-SD

Given an instance $\mathcal{I} = (G, \{d\}, J, w, h, k, Q)$, the min-max capacitated cycle cover problem with a single depot (CCCP-SD) is to determine a set $\mathcal{P}$ of k cycles so as to minimize cost($\mathcal{P}$), such that the k cycles cover every customer in J exactly once, and that each cycle starts and ends at a given depot d and has each of its trips containing at most Q customers. To develop an approximation algorithm for this problem, we revise step 4 of Algorithm 3.2 by joining the depot d to both the endpoints of each segment in $\mathcal{S}$, so that the resulting set $\mathcal{P}$ consists of exactly k cycles. Such a revised Algorithm 3.2 achieves an approximation ratio of $(4-1/k)$ for the min-max CCCP-SD as stated below.

Theorem 3.3. *The revised Algorithm 3.2 is a $(4-1/k)$-approximation algorithm for the min-max CCCP-SD.*

Proof. It is easy to see that the revised Algorithm 3.2 runs in polynomial time. To prove the approximation ratio, notice that for any cycle $\tilde{C}$ that contains the depot d, due to the triangle inequality, we have $w(d, v) \leq w(\tilde{C})/2$ for each customer v on $\tilde{C}$. Therefore, by following the derivation of (3.2) in the proof of Lemma 3.2, we obtain

$$\sum_{v \in J} \frac{w(d, v)}{Q'} \leq \frac{k}{2} \cdot \mathrm{OPT}(\mathcal{I}). \tag{3.13}$$

Moreover, it is easy to check the (3.3), (3.4), (3.5), and (3.6) remain valid for the

min-max CCCP-SD. Combining (3.3)–(3.6) and (3.13), we have

$$\begin{aligned}
\frac{w'(C)}{3k} &= \frac{w(C)+2h(J)}{3k} \le \frac{1}{3k}\Big[w(C')+2h(J)+2\sum_{v\in J}\frac{w(d,v)}{Q'}\Big] \\
&\le \frac{2}{3k}\Big[w(E(T^*))+h(J)+\frac{k}{2}\cdot\mathrm{OPT}(\mathcal{I})\Big] \\
&\le \frac{2}{3k}\Big[k\cdot\mathrm{OPT}(\mathcal{I})+\frac{k}{2}\cdot\mathrm{OPT}(\mathcal{I})\Big] = \mathrm{OPT}(\mathcal{I}),
\end{aligned} \tag{3.14}$$

where C is the cycle obtained from step 2 of the revised Algorithm 3.2. Following a similar argument as in the derivation of (2.7) for Theorem 2.1, and from the fact that step 4 of Algorithm 3.2 joins the depot d to both the endpoints of each segment in $\mathcal{S}$, we obtain

$$\mathrm{cost}(\mathcal{P}) \le \max\big\{h_{\max}+2w_{\max}(d),\, b+2w_{\max}(d)\big\}, \tag{3.15}$$

where $b=[w'(C)-2w_{\max}(d)]/k$. Note that $2w_{\max}(d)\le\mathrm{OPT}(\mathcal{I})$ (due to the triangle inequality) and that $h_{\max}\le\mathrm{OPT}(\mathcal{I})$. These, together with (3.14) and (3.15), imply that

$$\mathrm{cost}(\mathcal{P}) \le \max\Big\{\frac{w'(C)-2w_{\max}(d)}{k}+2w_{\max}(d),\, h_{\max}+2w_{\max}(d)\Big\} \le \Big(4-\frac{1}{k}\Big)\mathrm{OPT}(\mathcal{I}).$$

□

3.6.2 Min-max CPCP-SD-ST

As a variant of the min-max CPCP-SD, the min-max capacitated path cover problem with single trips (CPCP-SD-ST) forbids any replenishment after the vehicles have left their starting points. As a result, each path in a feasible solution must form a single trip that contains at most Q customers, which implies that each vehicle can service at most Q customers. Campbell et al. [15] have developed a heuristic for this problem but without proving any constant approximation ratio guarantee. In the

following, we present a revised version of Algorithm 2.2 and show that it has a worst-case approximation ratio of 7. Note that the problem is infeasible when $|J| > kQ$. Hence, we only consider the case where $|J| \leq kQ$.

Consider any instance $\mathcal{I} = (G, \{d\}, J, w, h, k, Q)$ of the min-max CPCP-SD-ST. The revised Algorithm 2.2 can be described as follows:

1. Follow step 1 of Algorithm 2.2 to obtain a cycle C that covers V and has no repeated vertices except for its endpoints.

2. Revise step 2 of Algorithm 2.2 as follows: Let $b = w'(C)/k$. Define thresholds $b_i := b$ for $1 \leq i \leq k-1$ and $b_k := w'(C) - \sum_{i=1}^{k-1} b_i$. Apply Split$(C, k, \{b_i\}_{1 \leq i \leq k})$ to split C into a set of k segments, denoted $\mathcal{S} = \{S_i : 1 \leq i \leq k\}$. Short-cut the depot so that every segment in $\mathcal{S}$ contains no depot.

3. Initialize the path set $\mathcal{P}$ to be an empty set. For $i = 1, 2, \ldots, k$, do the following:

   (a) Find $S_{\min}$ and $S_{\max}$, where $S_{\min}$ is a segment in $\mathcal{S}$ that contains the smallest number of customers, and $S_{\max}$ is a segment in $\mathcal{S}$ that contains the largest number of customers.

   (b) If $|J(S_{\max})| \leq Q$, then stop the iteration and go to step 4.

   (c) Otherwise, from any endpoint of $S_{\max}$, relabel the vertices on $S_{\max}$ as $\langle v_1, v_2, \ldots, v_{|J(S_{\max})|} \rangle$.

      i. Set $j := Q - |J(S_{\min})|$.

      ii. Let $S_{\max,j}$ denote the portion $\langle v_1, \ldots, v_j \rangle$ of $S_{\max}$, and let $\bar{S}_{\max,j}$ denote the portion $\langle v_{j+1}, \ldots, v_{|J(S_{\max}|)} \rangle$ of $S_{\max}$. Let u denote an endpoint of $S_{\min}$. Connect $S_{\min}$, $S_{\max,j}$, and d to form a path P_i by adding the edges (u, v_1) and (v_j, d). Relabel P_i so that its starting endpoint is at d. Add P_i to the path set $\mathcal{P}$.

      iii. Replace $S_{\max}$ in $\mathcal{S}$ with $\bar{S}_{\max,j}$. Remove $S_{\min}$ from $\mathcal{S}$.

4. For each segment remained in $\mathcal{S}$, if it contains at least one vertex, connect one of its endpoints to the depot d to form a path, which is then added to $\mathcal{P}$; otherwise, add a path $\langle d \rangle$ to $\mathcal{P}$. Return the path set $\mathcal{P}$.

It is easy to see that in step 2, $b_k \leq b$. Furthermore, the segments generated by step 2 contain no depot and contain each customer exactly once. For each iteration of step 3, if $|J(S_{\max})| \geq Q+1$, then in step 3(c) we "transfer" the first $Q - |J(S_{\min})|$ customers of segment $S_{\max}$ to segment $S_{\min}$, and then, after adding a depot to $S_{\min}$, we obtain a path with exactly Q customers. We can always guarantee that in step 3(c)(i), $|J(S_{\min})| \leq Q-1$ (and thus, $j \geq 1$). To see this, suppose the ith iteration is the first iteration where each segment in $\mathcal{S}$ contains at least Q customers, but segment $S_{\max}$ contains more than Q customers. At this moment, $\mathcal{S}$ contains $k-i+1$ segments, and $\mathcal{P}$ contains $i-1$ paths. Because each of these $i-1$ paths contains exactly Q customers, we have $|J| > kQ$, which violates the condition "$|J| \leq kQ$".

Theorem 3.4. *The revised Algorithm 2.2 is a 7-approximation algorithm for the min-max CPCP-SD-ST.*

Proof. Clearly, the revised Algorithm 2.2 runs in polynomial time. It is easy to see that the path set $\mathcal{P}$ returned by step 4 contains exactly k paths, that each path in $\mathcal{P}$ starts from a depot, and that segments in $\mathcal{S}$ obtained in step 2 cover all customers in J and so do the paths in $\mathcal{P}$. Moreover, as explained above, each path in $\mathcal{P}$ must contain at most Q customers. Therefore, $\mathcal{P}$ is a feasible solution to $\mathcal{I}$.

Next, we show that $\text{cost}(\mathcal{P}) \leq 7\text{OPT}(\mathcal{I})$. Following the same argument as in the proof of Lemma 2.3, we obtain that $\text{OPT}(\mathcal{I}) \geq \max\{h_{\max}, w_{\max}(d), b/2\}$, where $b = w'(C)/k$. By Lemma 2.2(v), we have $\text{cost}(\mathcal{S}) \leq \max\{w'(C)/k,\, h_{\max}\} \leq 2\text{OPT}(\mathcal{I})$. Consider the two segments $S_{\min}$ (with an endpoint u) and $S_{\max,j}$ (with endpoints v_1 and v_j), which are used to construct a path P_i in step 3(c) during the ith iteration. By the triangle inequality, we have $w(u, v_1) \leq w(u, d) + w(d, v_1) \leq 2w_{\max}(d)$. This,

together with the fact that $w(v_j, d) \leq w_{\max}(d)$, implies that $w(P_i)+h(P_i) \leq 2\text{cost}(\mathcal{S})+3w_{\max}(d)$ (because, according to step 3(c), P_i is formed by combining two segments with edges (u, v_1) and (v_j, d)). Thus, $w(P_i)+h(P_i) \leq 7\text{OPT}(\mathcal{I})$. In addition, for each path constructed in step 4, its total edge and vertex weight must not be greater than $\text{cost}(\mathcal{S}) + w_{\max}(d) \leq 3\text{OPT}(\mathcal{I})$. Hence, we conclude that $\text{cost}(\mathcal{P}) \leq 7\text{OPT}(\mathcal{I})$. □

3.7 Summary

In this chapter, we have developed the first constant ratio approximation algorithms for two variants of a min-max capacitated path cover problem, where a fleet of vehicles with limited capacity start from the same depot or from any depot in a given set of multiple depots to serve customers located in a metric undirected graph, so as to minimize the latest service completion time. Based on the results presented in this chapter, our future work will focus on improving the worst-case approximation ratios for problems studied in this chapter, or developing constant ratio approximation algorithms for other more complicated variants of the min-max capacitated path cover problem.

CHAPTER 4

Approximation Hardness for the Min-max Path Cover Problems

4.1 Introduction

In this chapter, we derive the first results on approximation hardness for the min-max UPCP-SD, min-max UPCP-MD, min-max CPCP-SD and the min-max CPCP-MD, which are defined in the previous two chapters, by showing that it is impossible for them to achieve approximation ratios less than 4/3, 3/2, 3/2, and 2, respectively, unless *P=NP*. The method for deriving approximation hardness results relies on a reduction from 3-Dimensional Matching (3DM) that is a well-known *NP*-complete problem. If there exist polynomial-time θ-approximation algorithms for these four variants of the min-max path cover problem, where θ is less than 4/3, 3/2, 3/2 and 2, respectively, we can show that these algorithms can be used to solve 3DM in polynomial time, which contradicts to the fact that 3DM is *NP*-complete.

The remainder of this chapter is organized as follows. Approximation hardness results for the the min-max UPCP-SD, min-max UPCP-MD, min-max CPCP-SD and the min-max CPCP-MD are proved in Section 4.2, Section 4.3, Section 4.4 and Section 4.5, respectively. In Section 4.6, we show a possible extension of our method to prove approximation hardness result for the min-max capacitated cycle cover problem

with a single depot.

4.2 Min-max UPCP-SD

The following theorem gives an approximation hardness result for the min-max UPCP-SD.

Theorem 4.1. *Unless P=NP, there is no polynomial-time* $(4/3-\epsilon)$*-approximation algorithm for the min-max UPCP-SD for any* $\epsilon > 0$*, even if* $h(v) = 0$ *for all* $v \in V$*.*

Proof. Consider the 3-dimensional matching (3DM) problem defined as follows:

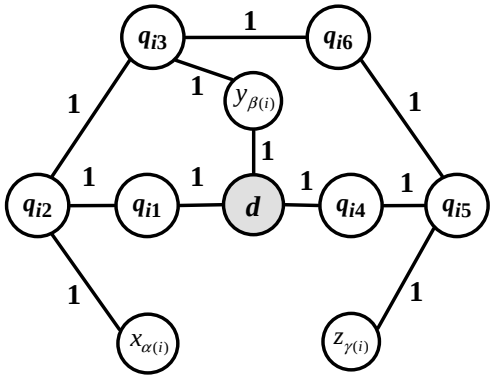
Instance: Set $\mathcal{M} \subseteq X \times Y \times Z$ with $|\mathcal{M}| = m$, where X, Y, and Z are disjoint sets having the same number n of elements.

Question: Does $\mathcal{M}$ contain a matching $\mathcal{M}' \subseteq \mathcal{M}$ such that $|\mathcal{M}'| = n$ and no two elements of $\mathcal{M}'$ agree in any coordinate?

It is well-known that 3DM is *NP*-complete [60]. Suppose, to the contrary, that there is an $\epsilon > 0$ such that a polynomial-time $(4/3-\epsilon)$-approximation algorithm exists for the min-max UPCP-SD with $h(v) = 0$ for all $v \in V$. Then, we will show that this algorithm can be used to solve 3DM in polynomial-time. We denote $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and $Z = \{z_1, z_2, \ldots, z_n\}$.

Given any instance of 3DM, we construct the corresponding instance of the min-max UPCP-SD $\mathcal{I} = (G, d, J, w, h, k, \infty)$ as follows: Let $G = (V, E)$ be a complete undirected graph with vertex set $V = \{d\} \cup X \cup Y \cup Z \cup \bigcup_{i=1}^{m} \{q_{ij} : 1 \leq j \leq 6\}$, where d is the depot. The customer set is $J = V \setminus \{d\}$. For each $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}$, we define $\tilde{E}_i$ as the edge subset consisting of the 11 edges as depicted in Figure 4.1, where $\alpha(i), \beta(i), \gamma(i) \in \{1, 2, \ldots, n\}$ and $i = 1, 2, \ldots, m$. (In Figure 4.1, the grey vertex represents the depot.) Let $\tilde{E} = \bigcup_{i=1}^{m} \tilde{E}_i$. We let $w(e) = 1$ for each $e \in \tilde{E}$, and

Figure 4.1: A component for $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)})$ in transforming 3DM to min-max UPCP-SD.



$w(e) = 2$ for each $e \in E \setminus \tilde{E}$. Clearly, the edge weight function w forms a metric. We let $h(v) = 0$ for all $v \in V$. Finally, we let $k = 2m + n$.

Define $\mathcal{F}(3, 3)$ as the set of paths P in G such that P starts from the depot d, that $|J(P)| = 3$, and that $w(P) \leq 3$. Then, for each path $P \in \mathcal{F}(3, 3)$, every edge of P must have an edge weight equal to 1. From Figure 4.1, it is easy to see that for each $i = 1, 2, \ldots, m$, there are six possible paths in $P \in \mathcal{F}(3, 3)$ that traverse the edges in $\tilde{E}_i$. They include:

$$P_{i1} = \langle d, q_{i1}, q_{i2}, x_{\alpha(i)} \rangle, \quad P_{i2} = \langle d, q_{i1}, q_{i2}, q_{i3} \rangle, \quad P_{i3} = \langle d, y_{\beta(i)}, q_{i3}, q_{i2} \rangle,$$
$$P_{i4} = \langle d, y_{\beta(i)}, q_{i3}, q_{i6} \rangle, \quad P_{i5} = \langle d, q_{i4}, q_{i5}, q_{i6} \rangle, \quad P_{i6} = \langle d, q_{i4}, q_{i5}, z_{\gamma(i)} \rangle.$$

Because all edge and vertex weights are integers, the $(4/3 - \epsilon)$-approximation algorithm must return a solution with an integer objective value. We consider two different cases.

Case 1: The $(4/3 - \epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value greater than or equal to 4. In this case, we will show that the given instance of 3DM does not contain a matching. By contradiction, suppose it contains a matching $\mathcal{M}'$. Then, we can construct the following solution to instance $\mathcal{I}$: For $i = 1, 2, \ldots, m$, if $(x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}'$, then we assign paths P_{i1}, P_{i4}, and

P_{i6} to three different vehicles; otherwise we assign paths P_{i2} and P_{i5} to two different vehicles. The total number of assigned vehicles is $3n + 2(m - n) = k$. Because $\mathcal{M}'$ is a matching, each of the vertices $x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n$ is covered by exactly one path. Hence, this solution is feasible. The objective value of this solution equals 3. This implies that the optimal solution value of $\mathcal{I}$ is no greater than 3, which in turn implies that the approximation algorithm must return a feasible solution with an objective value no greater than $(4/3 - \epsilon)(3) < 4$, which is a contradiction. Therefore, in this case the given instance of 3DM does not contain a matching.

Case 2: The $(4/3 - \epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value less than or equal to 3. Let $\mathcal{P}$ denote the set of all paths in this feasible solution. In this case, each path $P \in \mathcal{P}$ covers no more than three customers. Note that $|J| = 6m + 3n$, $k = 2m + n$, and the paths cover all the customers in J. Thus, $\mathcal{P} \subseteq \mathcal{F}(3, 3)$, and no two different paths in $\mathcal{P}$ cover the same customer in J. Hence, for each $j = 1, 2, \ldots, n$, there exists a unique and distinct path in $\mathcal{P} \subseteq \mathcal{F}(3, 3)$ that covers y_j. According to the construction of G and the definition of $\mathcal{F}(3, 3)$, the path in $\mathcal{P}$ that covers y_j must not contain any edge in $E \setminus \tilde{E}$. Thus, it must be either $P_{\sigma(j),3}$ or $P_{\sigma(j),4}$ with $\beta(\sigma(j)) = j$, for some $\sigma(j) \in \{1, 2, \ldots, m\}$. However, $P_{\sigma(j),3}$ cannot be in $\mathcal{P}$, because vertex $q_{\sigma(j),1}$ cannot be covered by any path in $\mathcal{F}(3, 3)$ that covers no customer in $J(P_{\sigma(j),3})$. Thus, $P_{\sigma(j),4} \in \mathcal{P}$. This also implies that $P_{\sigma(j),1}$ (resp. $P_{\sigma(j),6}$) is also in $\mathcal{P}$, since $P_{\sigma(j),1}$ (resp. $P_{\sigma(j),6}$) is the only possible path in $\mathcal{F}(3, 3)$ that covers vertex $q_{\sigma(j),1}$ (resp. $q_{\sigma(j),5}$) without covering any vertex that is already covered by $P_{\sigma(j),4}$. Therefore, among all the paths in $\mathcal{P}$, only $P_{\sigma(j),1}$ covers vertex $x_{\alpha(\sigma(j))}$, only $P_{\sigma(j),4}$ covers vertex $y_{\beta(\sigma(j))}$, and only $P_{\sigma(j),6}$ covers vertex $z_{\gamma(\sigma(j))}$. Let $\mathcal{M}' = \{(x_{\alpha(\sigma(j))}, y_{\beta(\sigma(j))}, z_{\gamma(\sigma(j))}) : j = 1, 2, \ldots, n\}$. Then, $|\mathcal{M}'| = n$, and no two elements of $\mathcal{M}'$ agree in any coordinate. This implies that $\mathcal{M}'$ is a matching in the given instance of 3DM.

Summarizing Cases 1 and 2, we conclude that the $(4/3 - \epsilon)$-approximation algo-

rithm, which has a polynomial running time, can be used to determine whether or not the given instance of 3DM contains a matching. This is impossible unless *P=NP*. The proof is completed. □

4.3 Min-max UPCP-MD

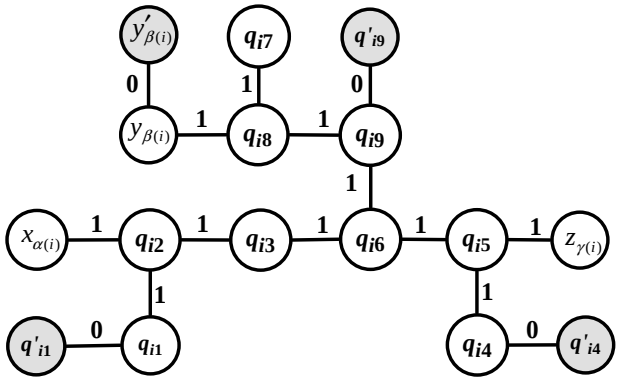
The following theorem gives an approximation hardness result for the min-max UPCP-MD.

Theorem 4.2. *Unless P=NP, there is no polynomial-time* $(3/2-\epsilon)$*-approximation algorithm for the min-max UPCP-MD for any* $\epsilon > 0$*, even if* $h(v) = 0$ *for all* $v \in V$*.*

Proof. Suppose, to the contrary, that there is an $\epsilon > 0$ such that a polynomial-time $(3/2-\epsilon)$-approximation algorithm exists for the min-max UPCP-MD with $h(v) = 0$ for all $v \in V$. Then, we will show that this algorithm can be used to solve 3DM in polynomial-time.

Consider any given instance $\mathcal{M} \subseteq X \times Y \times Z$ of 3DM with $|\mathcal{M}| = m$, where $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and $Z = \{z_1, z_2, \ldots, z_n\}$ are disjoint sets. We construct the corresponding instance $\mathcal{I} = (G, D, J, w, h, k, \infty)$ of the min-max UPCP-MD as follows: Let $G = (V, E)$ be a complete undirected graph with vertex set $V = D \cup J$, where $J = X \cup Y \cup Z \cup \bigcup_{i=1}^{m} \{q_{ij} : 1 \leq j \leq 9\}$ is the set of customers, and $D = \{y'_1, y'_2, \ldots, y'_n\} \cup \bigcup_{i=1}^{m} \{q'_{i1}, q'_{i4}, q'_{i9}\}$ is the set of depots. For each $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}$, we define $\tilde{E}_i$ as the edge subset consisting of the 15 edges as depicted in Figure 4.2, where $\alpha(i), \beta(i), \gamma(i) \in \{1, 2, \ldots, n\}$ and $i = 1, 2, \ldots, m$. (In Figure 4.2, the grey vertices are depots.) Let $\tilde{E} = \bigcup_{i=1}^{m} \tilde{E}_i$. In each $\tilde{E}_i$, let $w(e) = 0$ for each $e \in \{(y_{\beta(i)}, y'_{\beta(i)}), (q_{i1}, q'_{i1}), (q_{i4}, q'_{i4}), (q_{i9}, q'_{i9})\}$, and let $w(e) = 1$ for each $e \in \tilde{E}_i \setminus \{(y_{\beta(i)}, y'_{\beta(i)}), (q_{i1}, q'_{i1}), (q_{i4}, q'_{i4}), (q_{i9}, q'_{i9})\}$. For each $(u, v) \in E \setminus \tilde{E}$, we define the edge weight of (u, v) as the shortest distance from u to v in graph $(V, \tilde{E})$. Clearly, the edge weight function w forms a metric. We let $h(v) = 0$ for all $v \in V$. Finally, we let $k = 3m + n$.

Figure 4.2: A component for $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)})$ in transforming 3DM to min-max UPCP-MD.



Define $\mathcal{F}(3,2)$ as the set of paths P in G such that P starts from a depot in D, with $|J(P)| = 3$ and $w(P) \leq 2$. Then, for each path $P \in \mathcal{F}(3,2)$, the path P must have at least an edge with weight equal to 0, and it must not contain any edge with edge weight greater than 1. From Figure 4.2, it is easy to see that for each $i = 1, 2, \ldots, m$, the following seven paths are some of the possible paths in $\mathcal{F}(3,2)$ that traverse the edges in $\tilde{E}_i$:

$$P_{i1} = \langle q'_{i1}, q_{i1}, q_{i2}, x_{\alpha(i)} \rangle, \quad P_{i2} = \langle y'_{\beta(i)}, y_{\beta(i)}, q_{i8}, q_{i7} \rangle, \quad P_{i3} = \langle q'_{i4}, q_{i4}, q_{i5}, z_{\gamma(i)} \rangle,$$
$$P_{i4} = \langle q'_{i9}, q_{i9}, q_{i6}, q_{i3} \rangle, \quad P_{i5} = \langle q'_{i1}, q_{i1}, q_{i2}, q_{i3} \rangle, \quad P_{i5} = \langle q'_{i4}, q_{i4}, q_{i5}, q_{i6} \rangle,$$
$$P_{i7} = \langle q'_{i9}, q_{i9}, q_{i8}, q_{i7} \rangle.$$

Because all edge and vertex weights are integers, the $(3/2 - \epsilon)$-approximation algorithm must return a solution with an integer objective value. We consider two different cases.

Case 1: The $(3/2-\epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value greater than or equal to 3. In this case, we will show that the given instance of 3DM does not contain a matching. By contradiction, suppose it contains a matching $\mathcal{M}'$. Then, we can construct the following solution to instance $\mathcal{I}$: For $i = 1, 2, \ldots, m$, if $(x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}'$, then we assign paths P_{i1}, P_{i2}, P_{i3}, and P_{i4}

to four different vehicles; otherwise we assign paths P_{i5}, P_{i6}, and P_{i7} to three different vehicles. The total number of assigned vehicles is $4n + 3(m - n) = k$. Because $\mathcal{M}'$ is a matching, each of the vertices $x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n$ is covered by exactly one path. Hence, this solution is feasible. The objective value of this solution equals 2. This implies that the optimal solution value of $\mathcal{I}$ is no greater than 2, which in turn implies that the approximation algorithm must return a feasible solution with an objective value no greater than $(3/2 - \epsilon)(2) < 3$, which is a contradiction. Therefore, in this case the given instance of 3DM does not contain a matching.

Case 2: The $(3/2 - \epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value less than or equal to 2. Let $\mathcal{P}$ denote the set of all paths in this feasible solution. In this case, each path $P \in \mathcal{P}$ covers no more than three customers. Note that $|J| = 9m+3n$, $k = 3m+n$, and the paths cover all the customers in J. Thus, $\mathcal{P} \subseteq \mathcal{F}(3, 2)$, and no two different paths in $\mathcal{P}$ cover the same customer in J. Hence, for each $j = 1, 2, \ldots, n$, there exists a unique and distinct path in $\mathcal{P}$ that covers z_j. According to the construction of G and the definition of $\mathcal{F}(3, 2)$, the path in $\mathcal{P}$ that covers z_j must not contain any edge in $E \setminus \tilde{E}$. Thus, this path must be $P_{\sigma(j),3}$ for some $\sigma(j) \in \{1, 2, \ldots, m\}$ such that $\gamma(\sigma(j)) = j$. Since $P_{\sigma(j),4}$ is the only path in $\mathcal{F}(3, 2)$ which covers $q_{\sigma(j),6}$ without covering any customer of $P_{\sigma(j),3}$, we have $P_{\sigma(j),4} \in \mathcal{P}$. This also implies that $P_{\sigma(j),1}$ (resp. $P_{\sigma(j),2}$) is also in $\mathcal{P}$, because $P_{\sigma(j),1}$ (resp. $P_{\sigma(j),2}$) is the only possible path in $\mathcal{F}(3, 2)$ that covers $q_{\sigma(j),1}$ (resp. $q_{\sigma(j),7}$) without covering any customer that is already covered by $P_{\sigma(j),3}$ and $P_{\sigma(j),4}$. Therefore, among all the paths in $\mathcal{P}$, only $P_{\sigma(j),1}$ covers vertex $x_{\alpha(\sigma(j))}$, only $P_{\sigma(j),2}$ covers vertex $y_{\beta(\sigma(j))}$, and only $P_{\sigma(j),3}$ covers vertex $z_{\gamma(\sigma(j))}$. Let $\mathcal{M}' = \{(x_{\alpha(\sigma(j))}, y_{\beta(\sigma(j))}, z_{\gamma(\sigma(j))}) : j = 1, 2, \ldots, n\}$. Then, $|\mathcal{M}'| = n$, and no two elements of $\mathcal{M}'$ agree in any coordinate. This implies that $\mathcal{M}'$ is a matching in the given instance of 3DM.

Summarizing Cases 1 and 2, we conclude that the $(3/2 - \epsilon)$-approximation algorithm, which has a polynomial running time, can be used to determine whether or not

the given instance of 3DM contains a matching. This is impossible unless *P=NP*. □

4.4 Min-max CPCP-SD

The following theorem gives an approximation hardness result for the min-max CPCP-SD.

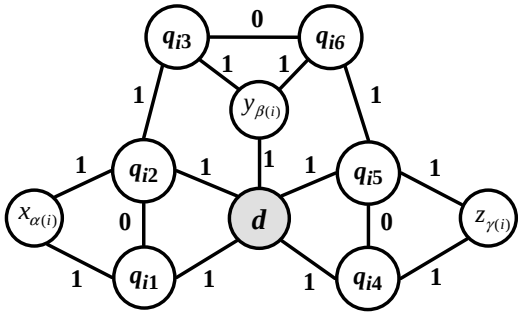
Theorem 4.3. *Unless P=NP, there is no polynomial-time* $(3/2-\epsilon)$*-approximation algorithm for the min-max CPCP-SD for any* $\epsilon > 0$*, even if* $h(v) = 0$ *for all* $v \in V$*.*

Proof. Suppose, to the contrary, that there is an $\epsilon > 0$ such that a polynomial-time $(3/2-\epsilon)$-approximation algorithm exists for the min-max CPCP-SD with $h(v) = 0$ for all $v \in V$. Then, we will show that this algorithm can be used to solve 3DM in polynomial-time.

Consider any given instance $\mathcal{M} \subseteq X \times Y \times Z$ of 3DM with $|\mathcal{M}| = m$, where $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and $Z = \{z_1, z_2, \ldots, z_n\}$ are disjoint sets. We construct the corresponding instance $\mathcal{I} = (G, \{d\}, J, w, h, k, Q)$ of the min-max CPCP-SD as follows: Let $G = (V, E)$ be a complete undirected graph with vertex set $V = \{d\} \cup X \cup Y \cup Z \cup \bigcup_{i=1}^{m} \{q_{ij} : 1 \leq j \leq 6\}$, where d is the depot. The customer set is $J = V \setminus \{d\}$. For each $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}$, we define $\tilde{E}_i$ as the edge subset consisting of the 16 edges as depicted in Figure 4.3, where $\alpha(i), \beta(i), \gamma(i) \in \{1, 2, \ldots, n\}$ and $i = 1, 2, \ldots, m$. Let $\tilde{E} = \bigcup_{i=1}^{m} \tilde{E}_i$. In each $\tilde{E}_i$, let $w(e) = 0$ for each $e \in \{(q_{i1}, q_{i2}), (q_{i4}, q_{i5}), (q_{i3}, q_{i6})\}$, and let $w(e) = 1$ for each $e \in \tilde{E}_i \setminus \{(q_{i1}, q_{i2}), (q_{i4}, q_{i5}), (q_{i3}, q_{i6})\}$. For each $(u, v) \in E \setminus \tilde{E}$, we define the edge weight of (u, v) as the shortest distance from u to v in graph $(V, \tilde{E})$. Clearly, the edge weight function w forms a metric. We let $h(v) = 0$ for all $v \in V$. Finally, we let $Q = 3$ and $k = 2m + n$.

Define $\mathcal{F}(3, 2)$ as the set of paths P in G such that P starts from the depot d, that $|J(P)| = 3$, and that $w(P) \leq 2$. Then, for each path $P \in \mathcal{F}(3, 2)$, every edge of

Figure 4.3: A component for $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)})$ in transforming 3DM to min-max CPCP-SD.



P must have an edge weight of at most 1, because there are no two adjacent edges in G with edge weights both equal to zero. From Figure 4.3, it is easy to see that for each $i = 1, 2, \ldots, m$, the following eight paths are some of the possible paths in $\mathcal{F}(3, 2)$ that traverse the edges in $\tilde{E}_i$:

$$P_{i1} = \langle d, q_{i1}, q_{i2}, x_{\alpha(i)} \rangle, \quad P_{i2} = \langle d, q_{i2}, q_{i1}, x_{\alpha(i)} \rangle, \quad P_{i3} = \langle d, y_{\beta(i)}, q_{i3}, q_{i6} \rangle,$$
$$P_{i4} = \langle d, y_{\beta(i)}, q_{i6}, q_{i3} \rangle, \quad P_{i5} = \langle d, q_{i4}, q_{i5}, z_{\gamma(i)} \rangle, \quad P_{i6} = \langle d, q_{i5}, q_{i4}, z_{\gamma(i)} \rangle,$$
$$P_{i7} = \langle d, q_{i1}, q_{i2}, q_{i3} \rangle, \quad P_{i8} = \langle d, q_{i4}, q_{i5}, q_{i6} \rangle.$$

Because all edge and vertex weights are integers, the $(3/2 - \epsilon)$-approximation algorithm must return a solution with an integer objective value. We consider two different cases.

Case 1: The $(3/2 - \epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value greater than or equal to 3. In this case, we will show that the given instance of 3DM does not contain a matching. By contradiction, suppose it contains a matching $\mathcal{M}'$. Then, we can construct the following solution to instance $\mathcal{I}$: For $i = 1, 2, \ldots, m$, if $(x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}'$, then we assign paths P_{i1}, P_{i3}, and P_{i5} to three different vehicles; otherwise we assign paths P_{i7} and P_{i8} to two different vehicles. The total number of assigned vehicles is $3n + 2(m - n) = k$. Because $\mathcal{M}'$ is a matching, each of the vertices $x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n$ is covered by exactly

one path. Hence, this solution is feasible. The objective value of this solution equals 2. This implies that the optimal solution value of $\mathcal{I}$ is no greater than 2, which in turn implies that the approximation algorithm must return a feasible solution with an objective value no greater than $(3/2-\epsilon)(2) < 3$, which is a contradiction. Therefore, in this case the given instance of 3DM does not contain a matching.

Case 2: The $(3/2-\epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value less than or equal to 2. Let $\mathcal{P}$ denote the set of all paths in this feasible solution. Each path $P \in \mathcal{P}$ contains at most one trip, since otherwise its total edge and vertex weight will exceed 2. Moreover, since $Q = 3$, each path $P \in \mathcal{P}$ covers no more than three customers. Note that $|J| = 6m + 3n$, $k = 2m + n$, and the paths cover all the customers in J. Thus, $\mathcal{P} \subseteq \mathcal{F}(3, 2)$, and no two different paths in $\mathcal{P}$ cover the same customer in J. Hence, for each $j = 1, 2, \ldots, n$, there exists a unique and distinct path in $\mathcal{P} \subseteq \mathcal{F}(3, 2)$ that covers y_j, which must be either $P_{\sigma(j),3}$ or $P_{\sigma(j),4}$ such that $\beta(\sigma(j)) = j$, for some $\sigma(j) \in \{1, 2, \ldots, m\}$. This also implies that $P_{\sigma(j),1}$ or $P_{\sigma(j),2}$ (resp. $P_{\sigma(j),5}$ or $P_{\sigma(j),6}$) is also in $\mathcal{P}$, since $P_{\sigma(j),1}$ and $P_{\sigma(j),2}$ (resp. $P_{\sigma(j),5}$ and $P_{\sigma(j),6}$) are the only possible paths in $\mathcal{F}(3, 2)$ that cover $q_{\sigma(j),1}$ (resp. $q_{\sigma(j),4}$) without covering any customer that is already covered by $P_{\sigma(j),1}$ or $P_{\sigma(j),2}$. Note that among all the paths in $\mathcal{P}$, only $P_{\sigma(j),1}$ or $P_{\sigma(j),2}$ covers vertex $x_{\alpha(\sigma(j))}$, only $P_{\sigma(j),3}$ or $P_{\sigma(j),4}$ covers vertex $y_{\beta(\sigma(j))}$, and only $P_{\sigma(j),5}$ or $P_{\sigma(j),6}$ covers vertex $z_{\gamma(\sigma(j))}$. Let $\mathcal{M}' = \{(x_{\alpha(\sigma(j))}, y_{\beta(\sigma(j))}, z_{\gamma(\sigma(j))}) : j = 1, 2, \ldots, n\}$. Then, $|\mathcal{M}'| = n$, and no two elements of $\mathcal{M}'$ agree in any coordinate. This implies that $\mathcal{M}'$ is a matching in the given instance of 3DM.

Summarizing Cases 1 and 2, we conclude that the $(3/2-\epsilon)$-approximation algorithm, which has a polynomial running time, can be used to determine whether or not the given instance of 3DM contains a matching. This is impossible unless *P=NP*. □

4.5 Min-max CPCP-MD

The following theorem gives an approximation hardness result for the min-max CPCP-MD.

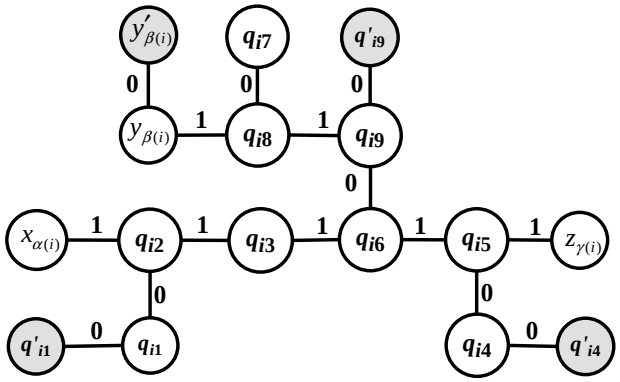
Theorem 4.4. *Unless P=NP, there is no polynomial-time* $(2-\epsilon)$*-approximation algorithm for the min-max CPCP-MD for any* $\epsilon > 0$*, even if* $h(v) = 0$ *for all* $v \in V$*.*

Proof. Suppose, to the contrary, that there is an $\epsilon > 0$ such that a polynomial-time $(2-\epsilon)$-approximation algorithm exists for the min-max CPCP-MD with $h(v) = 0$ for all $v \in V$. Then, we will show that this algorithm can be used to solve 3DM in polynomial-time.

Consider any given instance $\mathcal{M} \subseteq X \times Y \times Z$ of 3DM with $|\mathcal{M}| = m$, where $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and $Z = \{z_1, z_2, \ldots, z_n\}$ are disjoint sets. Similar to the proof of Theorem 4, we construct the corresponding instance $\mathcal{I} = (G, D, J, w, h, k, Q)$ of the min-max CPCP-MD as follows: Let $G = (V, E)$ be a complete undirected graph with vertex set $V = D \cup J$, where $J = X \cup Y \cup Z \cup \bigcup_{i=1}^{m} \{q_{ij} : 1 \leq j \leq 9\}$ is the set of customers, and $D = \{y'_1, y'_2, \ldots, y'_n\} \cup \bigcup_{i=1}^{m} \{q'_{i1}, q'_{i4}, q'_{i9}\}$ is the set of depots. For each $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}$, we define $\tilde{E}_i$ as the edge subset consisting of the 15 edges as depicted in Figure 4.4, where $\alpha(i), \beta(i), \gamma(i) \in \{1, 2, \ldots, n\}$ and $i = 1, 2, \ldots, m$. Let $\tilde{E} = \bigcup_{i=1}^{m} \tilde{E}_i$. In each $\tilde{E}_i$, let $w(e) = 0$ for each $e \in \{(y_{\beta(i)}, y'_{\beta(i)}), (q_{i1}, q'_{i1}), (q_{i4}, q'_{i4}), (q_{i9}, q'_{i9}), (q_{i1}, q_{i2}), (q_{i4}, q_{i5}), (q_{i7}, q_{i8}), (q_{i6}, q_{i9})\}$, and let $w(e) = 1$ for each $e \in \tilde{E}_i \setminus \{(y_{\beta(i)}, y'_{\beta(i)}), (q_{i1}, q'_{i1}), (q_{i4}, q'_{i4}), (q_{i9}, q'_{i9}), (q_{i1}, q_{i2}), (q_{i4}, q_{i5}), (q_{i7}, q_{i8}), (q_{i6}, q_{i9})\}$. For each $(u, v) \in E \setminus \tilde{E}$, we define the edge weight of (u, v) as the shortest distance from u to v in graph $(V, \tilde{E})$. Clearly, the edge weight function w forms a metric. We let $h(v) = 0$ for all $v \in V$. Finally, we let $Q = 3$ and $k = 3m + n$.

Define $\mathcal{F}(3, 1)$ as the set of paths P in G such that P starts from a depot in D, with $|J(P)| = 3$ and $w(P) \leq 1$. Then, for each path $P \in \mathcal{F}(3, 1)$, the path P must have at least two edges with weight equal to 0, and it must have no edge with

Figure 4.4: A component for $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)})$ in transforming 3DM to min-max CPCP-MD.



weight greater than 1. From Figure 4.4, it is easy to see that for each $i = 1, 2, \ldots, m$, the following eleven paths are some of the possible paths in $\mathcal{F}(3, 1)$ that traverse the edges in $\tilde{E}_i$:

$$P_{i1} = \langle q'_{i1}, q_{i1}, q_{i2}, x_{\alpha(i)} \rangle, \quad P_{i2} = \langle q'_{i1}, q_{i2}, q_{i1}, x_{\alpha(i)} \rangle, \quad P_{i3} = \langle y'_{\beta(i)}, y_{\beta(i)}, q_{i8}, q_{i7} \rangle,$$
$$P_{i4} = \langle y'_{\beta(i)}, y_{\beta(i)}, q_{i7}, q_{i8} \rangle, \quad P_{i5} = \langle q'_{i4}, q_{i4}, q_{i5}, z_{\gamma(i)} \rangle, \quad P_{i6} = \langle q'_{i4}, q_{i5}, q_{i4}, z_{\gamma(i)} \rangle,$$
$$P_{i7} = \langle q'_{i9}, q_{i9}, q_{i6}, q_{i3} \rangle, \quad P_{i8} = \langle q'_{i9}, q_{i6}, q_{i9}, q_{i3} \rangle, \quad P_{i9} = \langle q'_{i1}, q_{i1}, q_{i2}, q_{i3} \rangle,$$
$$P_{i,10} = \langle q'_{i4}, q_{i4}, q_{i5}, q_{i6} \rangle, \quad P_{i,11} = \langle q'_{i9}, q_{i9}, q_{i8}, q_{i7} \rangle.$$

Because all edge and vertex weights are integers, the $(2-\epsilon)$-approximation algorithm must return a solution with an integer objective value. We consider two different cases.

Case 1: The $(2-\epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value greater than or equal to 2. In this case, we will show that the given instance of 3DM does not contain a matching. By contradiction, suppose it contains a matching $\mathcal{M}'$. We can construct the following solution to instance $\mathcal{I}$: For $i = 1, 2, \ldots, m$, if $(x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}'$, then we assign paths P_{i1}, P_{i3}, P_{i5}, and P_{i7} to four different vehicles; otherwise we assign paths P_{i9}, $P_{i,10}$, and $P_{i,11}$ to three different vehicles. Following the same argument as in the proof of Theorem 4.2, this

solution is feasible with an objective value equal to 1. This implies that the optimal solution value of $\mathcal{I}$ is no greater than 1, which in turn implies that the approximation algorithm must return a feasible solution with an objective value no greater than $(2-\epsilon)(1) < 2$, which is a contradiction. Therefore, in this case the given instance of 3DM does not contain a matching.

Case 2: The $(2-\epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value less than or equal to 1. Let $\mathcal{P}$ denote the set of all paths in this feasible solution. In this case, it is not difficult to check that because $Q = 3$, each path $P \in \mathcal{P}$ covers no more than three customers. Note that $|J| = 9m + 3n$, $k = 3m + n$, and the paths cover all the customers in J. Thus, $\mathcal{P} \subseteq \mathcal{F}(3, 1)$, and no two different paths in $\mathcal{P}$ cover the same customer in J. Hence, for each $j = 1, 2, \ldots, n$, there exists a unique and distinct path in $\mathcal{P}$ that covers z_j, which must be either $P_{\sigma(j),5}$ or $P_{\sigma(j),6}$ such that $\beta(\sigma(j)) = j$, for some $\sigma(j) \in \{1, 2, \ldots, m\}$. This implies that $P_{\sigma(j),7}$ or $P_{\sigma(j),8}$ is also in $\mathcal{P}$, since $P_{\sigma(j),7}$ and $P_{\sigma(j),8}$ are the only possible paths in $\mathcal{F}(3, 1)$ which cover $q_{\sigma(j),6}$ without covering any customer of $P_{\sigma(j),5}$ or $P_{\sigma(j),6}$. This also implies that $P_{\sigma(j),1}$ or $P_{\sigma(j),2}$ (resp. $P_{\sigma(j),3}$ or $P_{\sigma(j),4}$) is also in $\mathcal{P}$, because $P_{\sigma(j),1}$ and $P_{\sigma(j),2}$ (resp. $P_{\sigma(j),3}$ and $P_{\sigma(j),4}$) are the only possible paths in $\mathcal{F}(3, 2)$ that cover $q_{\sigma(j),1}$ (resp. $q_{\sigma(j),7}$) without covering any customer that is already covered. Note that among all the paths in $\mathcal{P}$, only $P_{\sigma(j),1}$ or $P_{\sigma(j),2}$ covers vertex $x_{\alpha(\sigma(j))}$, only $P_{\sigma(j),3}$ or $P_{\sigma(j),4}$ covers vertex $y_{\beta(\sigma(j))}$, and only $P_{\sigma(j),5}$ or $P_{\sigma(j),6}$ covers vertex $z_{\gamma(\sigma(j))}$. Let $\mathcal{M}' = \{(x_{\alpha(\sigma(j))}, y_{\beta(\sigma(j))}, z_{\gamma(\sigma(j))}) : j = 1, 2, \ldots, n\}$. Then, $|\mathcal{M}'| = n$, and no two elements of $\mathcal{M}'$ agree in any coordinate. This implies that $\mathcal{M}'$ is a matching in the given instance of 3DM.

Summarizing Cases 1 and 2, we conclude that the $(2-\epsilon)$-approximation algorithm, which has a polynomial running time, can be used to determine whether or not the given instance of 3DM contains a matching. This is impossible unless *P=NP*. □

4.6 Min-max CCCP-SD

The following theorem states the approximation hardness of the min-max capacitated cycle cover problem with a single depot (CCCP-SD).

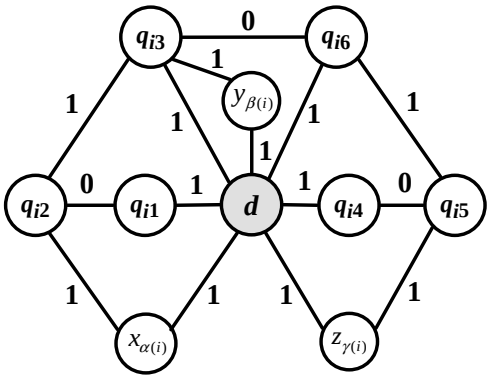
Theorem 4.5. *Unless P=NP, there is no polynomial-time* $(4/3-\epsilon)$*-approximation algorithm for the min-max CCCP-SD for any* $\epsilon > 0$*, even if* $h(v) = 0$ *for all* $v \in V$*.*

Proof. Suppose, to the contrary, that there is an $\epsilon > 0$ such that a polynomial-time $(4/3-\epsilon)$-approximation algorithm exists for the min-max CCCP-SD with $h(v) = 0$ for all $v \in V$. Then, we will show that this algorithm can be used to solve 3DM in polynomial-time.

Given any instance of 3DM, similar to the proof of Theorem 4.1, we construct the corresponding instance $\mathcal{I} = (G, \{d\}, J, w, h, k, Q)$ of the min-max CCCP-SD as follows: Let $G = (V, E)$ be a complete undirected graph with vertex set $V = \{d\} \cup X \cup Y \cup Z \cup \bigcup_{i=1}^{m} \{q_{ij} : 1 \leq j \leq 6\}$. For each $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}$, we define $\tilde{E}_i$ as the edge subset consisting of the 15 edges as depicted in Figure 4.5, where $\alpha(i), \beta(i), \gamma(i) \in \{1, 2, \ldots, n\}$ and $i = 1, 2, \ldots, m$. We define $w(e) = 0$ for each $e \in \{(q_{i1}, q_{i2}), (q_{i3}, q_{i6}), (q_{i4}, q_{i5})\}$, and $w(e) = 1$ for each $e \in \tilde{E}_i \backslash \{(q_{i1}, q_{i2}), (q_{i3}, q_{i6}), (q_{i4}, q_{i5})\}$. Let $\tilde{E} = \bigcup_{i=1}^{m} \tilde{E}_i$. For each $(u, v) \in E \setminus \tilde{E}$, we define the edge weight of (u, v) as the shortest distance from u to v in graph $(V, \tilde{E})$. Clearly, the edge weight function w forms a metric. We let $h(v) = 0$ for all $v \in V$. Finally, we let $Q = 3$ and $k = 2m + n$.

Define $\mathcal{F}(3, 3)$ as the set of cycles P in G such that P starts from the depot d and returns to d, that $|J(P)| = 3$, and that $w(P) \leq 3$. Then, for each cycle $P \in \mathcal{F}(3, 3)$, every edge of P must have an edge weight no greater than 1. From Figure 4.5, it is easy to see that for each $i = 1, 2, \ldots, m$, the following eight cycles are some of the

Figure 4.5: A component for $M_i = (x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)})$ in transforming 3DM to min-max CCCP-SD.



possible cycles in $\mathcal{F}(3,3)$ that traverse the edges in $\tilde{E}_i$:

$$P_{i1} = \langle d, q_{i1}, q_{i2}, x_{\alpha(i)}, d\rangle, \quad P_{i2} = \langle d, q_{i2}, q_{i1}, x_{\alpha(i)}, d\rangle, \quad P_{i3} = \langle d, y_{\beta(i)}, q_{i3}, q_{i6}, d\rangle,$$
$$P_{i4} = \langle d, y_{\beta(i)}, q_{i6}, q_{i3}, d\rangle, \quad P_{i5} = \langle d, q_{i4}, q_{i5}, z_{\gamma(i)}, d\rangle, \quad P_{i6} = \langle d, q_{i5}, q_{i4}, z_{\gamma(i)}, d\rangle,$$
$$P_{i7} = \langle d, q_{i1}, q_{i2}, q_{i3}, d\rangle, \quad P_{i8} = \langle d, q_{i4}, q_{i5}, q_{i6}, d\rangle.$$

We consider two possible cases.

Case 1: The $(4/3 - \epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value greater than or equal to 4. In this case, we will show that the given instance of 3DM does not contain a matching. By contradiction, suppose it contains a matching $\mathcal{M}'$. Then, we can construct the following solution to instance $\mathcal{I}$: For $i = 1, 2, \ldots, m$, if $(x_{\alpha(i)}, y_{\beta(i)}, z_{\gamma(i)}) \in \mathcal{M}'$, then we assign cycles P_{i1}, P_{i3}, and P_{i5} to three different vehicles; otherwise we assign cycles P_{i7} and P_{i8} to two different vehicles. The total number of assigned vehicles is $3n + 2(m - n) = k$. Because $\mathcal{M}'$ is a matching, each of the vertices $x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n$ is covered by exactly one cycle. Hence, this solution is feasible. The objective value of this solution equals 3. This implies that the optimal solution value of $\mathcal{I}$ is no greater than 3, which in turn implies that the approximation algorithm must return a feasible solution with an objective value no greater than $(4/3 - \epsilon)(3) < 4$, which is a contradiction. Therefore,

in this case the given instance of 3DM does not contain a matching.

Case 2: The $(4/3-\epsilon)$-approximation algorithm returns a feasible solution to $\mathcal{I}$ with an objective value less than or equal to 3. Let $\mathcal{P}$ denote the set of all cycles in this feasible solution. By a similar argument as in the proof of Theorem 4.1, $\mathcal{P} \subseteq \mathcal{F}(3,3)$, and no two different cycles in $\mathcal{P}$ cover the same customer in J. Hence, for each $j = 1, 2, \ldots, n$, there exists a unique and distinct cycle in $\mathcal{P} \subseteq \mathcal{F}(3,3)$ that covers y_j. Thus, it must be either $P_{\sigma(j),3}$ or $P_{\sigma(j),4}$ (or the reverses of these two cycles) with $\beta(\sigma(j)) = j$, for some $\sigma(j) \in \{1, 2, \ldots, m\}$. This also implies that $P_{\sigma(j),1}$ or $P_{\sigma(j),2}$ (resp. $P_{\sigma(j),5}$ or $P_{\sigma(j),6}$) is also in $\mathcal{P}$. Note that among all the cycles in $\mathcal{P}$, only $P_{\sigma(j),1}$ or $P_{\sigma(j),2}$ covers vertex $x_{\alpha(\sigma(j))}$, only $P_{\sigma(j),3}$ or $P_{\sigma(j),4}$ covers vertex $y_{\beta(\sigma(j))}$, and only $P_{\sigma(j),5}$ or $P_{\sigma(j),6}$ covers vertex $z_{\gamma(\sigma(j))}$. Let $\mathcal{M}' = \{(x_{\alpha(\sigma(j))}, y_{\beta(\sigma(j))}, z_{\gamma(\sigma(j))}) : j = 1, 2, \ldots, n\}$. Then, $|\mathcal{M}'| = n$, and no two elements of $\mathcal{M}'$ agree in any coordinate. This implies that $\mathcal{M}'$ is a matching in the given instance of 3DM.

Summarizing Cases 1 and 2, we conclude that the $(4/3-\epsilon)$-approximation algorithm, which has a polynomial running time, can be used to determine whether or not the given instance of 3DM contains a matching. This is impossible unless *P=NP*. □

4.7 Summary

In this chapter, we have proved the first approximation hardness results for four typical variants of the min-max path cover problem. Furthermore, we have demonstrated an extension of our method to derive the first approximation hardness result for the min-max capacitated cycle cover problem with a single depot. Based on the results presented in this chapter, our future work will focus on improving the approximation hardness results for problems studied in this chapter, or deriving approximation hardness results for other min-max vehicle routing problems.

CHAPTER 5

Approximation Schemes for the Min-max 2-TSPT and Its Extensions

5.1 Introduction

The min-max 2-Traveling Salesmen Problem on a Tree (2-TSPT) aims to determine a set of two tours for two vehicles to serve customers located in a tree, such that each of the two tours starts from and returns to a given depot, with the maximum length of the two tours minimized. Although the min-max 2-TSPT has been proved to be *NP*-hard by Averbakh and Berman [8], it is not clear whether or not it is *NP*-hard in the strong sense, and this has been an open question for a decade. Motivated by this open question, we investigate in this chapter as to whether there exists a pseudo-polynomial time exact algorithm, or even a fully polynomial time approximation scheme (FPTAS), for the min-max 2-TSPT.

The remainder of this chapter is organized as follows. Section 5.2 introduces the definition of the min-max 2-TSPT and the notation that are used throughout this chapter. Section 5.3 illustrates the main results presenting a pseudo-polynomial time exact algorithm and an FPTAS for the min-max 2-TSPT. In Section 5.4, we generalize these algorithms for several multiple-vehicle routing problems in trees, including the min-max k-TSPT, which is an extension of the min-max 2-TSPT by taking into

consideration k vehicles. This chapter is then summarized in Section 5.5.

5.2 Notation and Problem Definition

Consider a tree $T = (V, E)$ with the vertex set denoted by $V = \{1, 2, ..., n\}$ and the edge set denoted by E, where $s \in V$ denotes the root and the depot of T. For each edge $(u, v) \in E$, it has a non-negative integer edge weight denoted by $w(u, v)$. A vertex u is called an ancestor of vertex v if u lies on the unique path from the root s to v; a vertex v is called a descendant of vertex u if u is an ancestor of vertex v. For each vertex $v \in V$, the depth of v in T is defined as the number of edges on the path from the root s to v. The min-max 2-TSPT is to decide two tours that start from s and return to s, so as to visit all the vertices in V through the edges of E with the maximum total edge weight of the two tours minimized. Thus, we use $[T, w, 2]$ to denote an instance of the min-max 2-TSPT.

Let $\Phi = (\emptyset, \emptyset)$ denote an empty tree whose vertex and edge sets are both empty. For any subtree Q of T, let $V(Q)$ and $E(Q)$ denote the vertex set and the edge set of Q, respectively. Thus, the subtree Q can be denoted as $(V(Q), E(Q))$. Let $w(Q)$ denote the total edge weight of Q. For any edge subset $E' \subseteq E$, we use $w(E')$ to denote the total edge weight of E'. Moreover, let $W = w(T)$ denote the total edge weight of T.

For each tour in T, the subtree of T that spans the vertices of the tour has a weight of at most half of the tour. For each subtree of T, duplicating the edges of the subtree constitutes a tour in T with a weight of at most double of the subtree. Thus, the min-max 2-TSPT can be equivalently defined so as to determine a two-subtree tuple (Q_1, Q_2) of T, such that $V(Q_1) \cup V(Q_2)$ covers all vertices in V, and that both $V(Q_1)$ and $V(Q_2)$ cover s, with $\max\{w(Q_1), w(Q_2)\}$ minimized:

MIN-MAX 2-TSPT

Instance: $[T, w, 2]$, $T = (V, E)$ is a tree rooted at s where $V = \{1, 2, ..., n\}$, and $w : E \to \mathbb{Z}^+$ represents edge weights.

Feasible solution: A two-subtree tuple (Q_1, Q_2) of T such that $V \subseteq V(Q_1) \bigcup V(Q_2)$, and $s \in V(Q_i)$, for $i = 1, 2$.

Objective: To minimize $\max\{w(Q_1), w(Q_2)\}$.

Given an instance $[T, w, 2]$ of the min-max 2-TSPT, we use $(\widetilde{T}_1, \widetilde{T}_2)$ to denote an optimum solution to $[T, w, 2]$, and let OPT $= \max\{w(\widetilde{T}_1), w(\widetilde{T}_2)\}$ indicate the optimal objective value.

5.3 Main Results

In this section, we first introduce a transformation in Section 5.3.1 that can transform any instance of the min-max 2-TSPT defined on a general tree to a "standard" instance defined on a full binary tree, so as to simplify the presentation of our main results for the min-max 2-TSPT. The main results for the min-max 2-TSPT include a pseudo-polynomial time exact algorithm in Section 5.3.2 and an FPTAS in Section 5.3.3.

5.3.1 Transformation to standard instances

We define the "standard" instances of the min-max 2-TSPT as follows.

Definition 5.1. An instance of the min-max 2-TSPT is standard if and only if the underlying tree is a full binary tree, which means that each vertex in the tree other than the leaves has exactly two children.

Given two instances $[T, w, 2]$ and $[T', w', 2]$ of the min-max 2-TSPT, if any feasible solution to $[T, w, 2]$ can be transformed in polynomial time to a feasible solution to $[T', w', 2]$ with the same objective value, and vice versa, then the two instances are

defined as being equivalent. Accordingly, if there exists a pseudo-polynomial time exact algorithm (or respectively, an FPTAS) for $[T', w', 2]$, then the same algorithm can be applied to obtain a pseudo-polynomial time exact algorithm (or respectively, an FPTAS) for $[T, w, 2]$, and vice versa.

Given any instance $[T, w, 2]$ of the min-max 2-TSPT, we can transform it to a standard instance $[T', w', 2]$ by Algorithm 5.1.

Algorithm 5.1.

Input: An instance $[T, w, 2]$

Output: A standard instance $[T', w', 2]$.

1. Set $T' = T$ and $w' = w$.

2. Repeat the following steps until no vertex in T' has only one child:

   (a) Let v denote any vertex in T' that has only one child.

   (b) Insert a new vertex $v' = |V(T')| + 1$ into T' to be the second child of v by adding an edge (v, v') and setting $w'(v, v') = 0$

3. Repeat the following steps until no vertex in T' has more than two children:

   (a) Let v denote any vertex in T' that has more than two children.

   (b) Add a new vertex $v' = |V(T')| + 1$. For each child u of v, other than the first child, move u to be a child of v' by replacing the edge (u, v) with an edge (u, v') and setting $w'(u, v) = w'(u, v')$.

   (c) Insert v' into T' to be the second child of v by adding an edge (v, v') and setting $w'(v, v') = 0$.

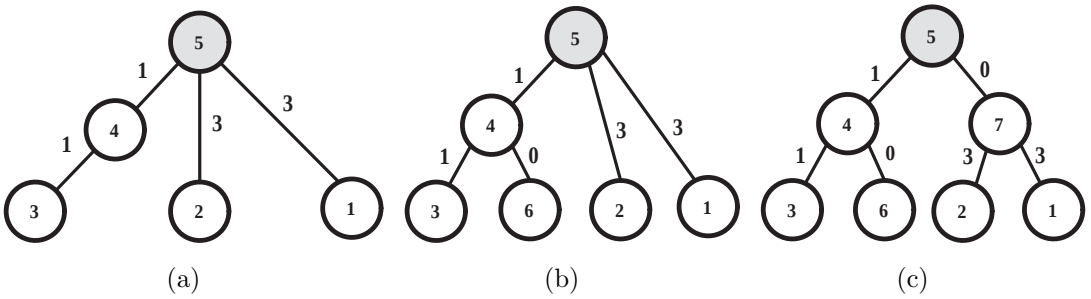
4. Return the resulting instance $[T', w', 2]$.

□

We use Example 5.1 to illustrate Algorithm 5.1.

Example 5.1. Consider an instance $[T, w, 2]$ shown in Figure 5.1(a), where the tree T is rooted at vertex 5, and the edge weight $w(e)$ is indicated by an integer on each edge $e \in E(T)$.

Figure 5.1: Example to illustrate Algorithm 5.1.



Algorithm 5.1 first sets $T' = T$ and $w' = w$ in Step 1. Since vertex 4 in T' has only one child, as depicted in Figure 5.1(b), Step 2.b inserts a new vertex 6 into T' to be the second child of vertex 4 by adding an edge $(4, 6)$ and setting $w'(4, 6) = 0$. Since no vertex in T' has only one child, then Algorithm 5.1 goes to Step 3. Since vertex 5 has more than two children, Step 3.b adds a new vertex 7, moves vertex 2 to be a child of vertex 7 by replacing the edge $(5, 2)$ with an edge $(7, 2)$ and setting $w'(7, 2) = w'(5, 2)$, and moves vertex 1 to be a child of vertex 7 by replacing the edge $(5, 1)$ with an edge $(7, 1)$ and setting $w'(7, 1) = w'(5, 1)$. It is noted that vertex 4 is still the first child of vertex 5. Then, Step 3.c inserts the new vertex 7 into T' to be the second child of 5 by adding an edge $(5, 7)$ and setting $w'(5, 7) = 0$. Since no vertex in T' has more than two children, the resulting instance $[T', w', 2]$, which is shown in Figure 5.1(c), is returned in Step 3. □

The following theorem states that any instance $[T, w, 2]$ to the min-max 2-TSPT can be transformed to an equivalent standard instance $[T', w', 2]$ by Algorithm 5.1.

Theorem 5.1. *Given any instance $[T, w, 2]$ of the min-max 2-TSPT, Algorithm 5.1 outputs an equivalent standard instance $[T', w', 2]$ in $O(n)$ time.*

Proof. In Algorithm 5.1, Step 2 terminates in $O(n)$ iterations, because each new vertex inserted in Step 2.b has no child. In each iteration of Step 3, let q denote the total number of non-leaf vertices in T' and let p denote the sum of the numbers of children for all non-leaf vertices. It can be seen from Step 3.b and Step 3.c that the value of $(p-2q)$ is decreasing after each iteration in Step 3. Since $(p-2q)$ is less than or equal to n before Step 3, and equals zero when Step 3 terminates, we obtain that Step 3 stops in $O(n)$ iterations. Thus, we obtain that the running time of Algorithm 5.1 is $O(n)$.

It can be seen that after Step 2 of Algorithm 5.1, no vertex in T' has only one child, and after Step 3, no vertex in T' has more than two children, or has only one child. Therefore, the resulting tree T' is a full binary tree, which, by definition, indicates that $[T', w', 2]$ is a standard instance. In the following, we only need to verify that $[T, w, 2]$ and $[T', w', 2]$ are equivalent.

On the one hand, given any feasible solution (Q_1, Q_2) to $[T, w, 2]$, for $i = 1, 2$, we can transform Q_i to a subtree Q'_i of T' by the following three steps:

Step 1: Set V_i initially to be $V(Q_i)$.

Step 2: For each $v \in V(T') \setminus V(T)$, let $a(v)$ denote the first ancestor of v in T' that is also in $V(T)$, and let $A(v)$ denote all the vertices lying on the path from $a(v)$ to v (including $a(v)$ and v). If $a(v) \in V_i$, add $A(v)$ to V_i.

Step 3: Return Q'_i as the subtree of T' induced by V_i.

It is easy to verify that the subtree Q'_i constructed above for $i = 1, 2$ is a connected subgraph, and satisfies that $V(Q_i) \subseteq V(Q'_i)$, and that $w'(Q'_i) = w(Q_i)$ according to the definition of w' in Step 2.b and Step 3.c of Algorithm 5.1. Therefore, we obtain that $\max\{w'(Q'_1), w'(Q'_2)\} = \max\{w(Q_1), w(Q_2)\}$. Since (Q_1, Q_2) is a feasible solution to $[T, w, 2]$, we have $V(T) = V(Q_1) \bigcup V(Q_2)$, and the root $s \in V(Q_i)$, for $i = 1, 2$. Since $V(Q_i) \subseteq V(Q'_i)$ for $i = 1, 2$, and since $V(T') \setminus V(T) \subseteq V(Q'_1) \bigcup V(Q'_2)$,

we obtain that $s \in V(Q'_i)$ for $i = 1, 2$, and $V(T') \subseteq V(Q'_1) \bigcup V(Q'_2)$. Thus, (Q'_1, Q'_2) is a feasible solution to $[T', w', 2]$ with $\max\{w'(Q'_1), w'(Q'_2)\} = \max\{w(Q_1), w(Q_2)\}$.

On the other hand, given any feasible solution (Q'_1, Q'_2) to $[T', w', 2]$, for $i = 1, 2$, we can transform Q'_i to a subtree Q_i of T by the following two steps:

Step 1: Label all vertices in $V(Q'_i) \backslash V(T)$ in a breadth-first order in Q'_i as $v_1, v_2, \ldots, v_{m_i}$.

Step 2: For $j = m_i, m_i - 1, \ldots, 1$, let $p(v_j)$ denote the parent of v_j in Q'_i, delete v_j and the edge $(p(v_j), v_j)$, and move each child u of v_j to be a child of $p(v_j)$ by replacing edge (v_j, u) with an edge $(p(v_j), u)$ and setting $w'(p(v_j), u) = w'(v_j, u)$.

It is easy to verify that the subtree Q_i constructed above for $i = 1, 2$ is a connected subgraph, and satisfies that $V(Q_i) \subseteq V(Q'_i)$, and that $w(Q_i) = w'(Q'_i)$ according to the definition of w' in Step 2.b and Step 3.c of Algorithm 5.1. Since (Q'_1, Q'_2) is a feasible solution to $[T', w', 2]$, we have $V(T') = V(Q'_1) \bigcup V(Q'_2)$, and the root $s \in V(Q'_i)$, for $i = 1, 2$. According to the construction of Q_i for $i = 1, 2$, all vertices in $V(Q'_i) \bigcap V(T)$ are maintained in Q_i, including the root s. We obtain that $s \in V(Q_i)$ for $i = 1, 2$, and that $V(T) \subseteq V(Q_1) \bigcup V(Q_2)$. Thus, since $V(Q_i) \subseteq V(T)$ for $i = 1, 2$, then (Q_1, Q_2) is a feasible solution to $[T, w, 2]$ with $\max\{w(Q_1), w(Q_2)\} = \max\{w'(Q'_1), w'(Q'_2)\}$. Hence, Theorem 5.1 is proved. □

5.3.2 Dynamic programming for the min-max 2-TSPT

Due to Theorem 5.1, we can assume without loss of generality that any given instance $[T, w, 2]$ of the min-max 2-TSPT is standard. Furthermore, to ease the presentation, we can assume without loss of generality that vertices in $V = \{1, 2, ..., n\}$ are labeled in a non-increasing order on their depths in T, so that for each $u, v \in V$, if v is a child of u, then $v \leq u - 1$. Finally, we assume without loss of generality that n is the root and the depot of T.

For each $v \in V$, let $T^{(v)}$ denote a subtree of T that contains v and all the decedents

of v. Let $\mathcal{S}^{(v)}$ denote a set of all the subtrees of $T^{(v)}$. For each subtree $Q \in \mathcal{S}^{(v)}$, recall that $w(Q)$ denotes the total edge weight of Q. We let a single bit $\alpha^{(v)}(Q)$ equal 1 if Q contains v, and equal 0 otherwise. Let a single bit $\beta(Q)$ equal 1 if Q is an empty tree, and equal 0 otherwise. Thus, we use a 3-tuple $(w(Q), \alpha^{(v)}(Q), \beta(Q))$ to define the state of Q as a subtree of $T^{(v)}$.

To develop an exact algorithm for the min-max 2-TSPT, we first define Procedure 5.1 as follows, which, for any $v \in V$, and for any subtree L of $T^{(l)}$ and any subtree R of $T^{(r)}$, where l and r are the left and right children of v respectively, joins L and R to form subtrees of $T^{(v)}$.

Procedure 5.1.

Input: A vertex $v \in V$, a subtree L of $T^{(l)}$ where l is the left child of v, and a subtree R of $T^{(r)}$ where r is the right child of v.

Output: A set $\mathcal{Q}$ of all the subtrees of $T^{(v)}$ that contain L as a subtree induced by $V(T^{(l)})$, and contain R as a subtree induced by $V(T^{(r)})$.

1. Define $G_1 = \Phi$, $G_2 = (\{v\}, \emptyset)$, $G_3 = (\{v, l\}, \{(v, l)\})$, $G_4 = (\{v, r\}, \{(v, r)\})$, and $G_5 = (\{v, l, r\}, \{(v, l), (v, r)\})$.

2. Set $\mathcal{Q} = \emptyset$. Then, for $t = 1, 2, .., 5$, do the following steps:

   (a) Let $T_t = (L \cup R) \cup G_t$.

   (b) If the following three conditions are satisfied, then add T_t to $\mathcal{Q}$.

      (i) T_t is connected.

      (ii) The subgraph of T_t induced by $V(T^{(l)})$ is equal to L.

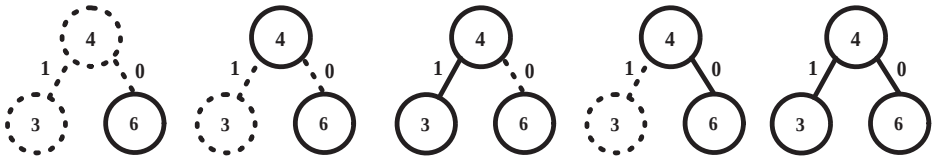
      (iii) The subgraph of T_t induced by $V(T^{(r)})$ is equal to R.

3. Return $\mathcal{Q}$. □

We use Example 5.2 to illustrate Procedure 5.1.

Example 5.2. Consider the vertex $v = 4$ of the full binary tree in the standard instance shown in Figure 5.1(c), which has exactly two children, where vertex 3 is the left child and vertex 6 is the right child. Given a subtree $L = \Phi$ of $T^{(3)}$ and a subtree $R = (\{6\}, \emptyset)$ of $T^{(6)}$, let us apply Procedure 5.1 on v, L, and R to obtain a set $\mathcal{Q}$ of subtrees of $T^{(4)}$. According to the definition of G_t, for $t = 1, 2, \ldots, 5$, in Step 1, we have $G_1 = \Phi$, $G_2 = (\{4\}, \emptyset)$, $G_3 = (\{4, 3\}, \{(4, 3)\})$, $G_4 = (\{4, 6\}, \{(4, 6)\})$, and $G_5 = (\{4, 3, 6\}, \{(4, 3), (4, 6)\})$. Accordingly, we can obtain $T_t = (L \cup R) \cup G_t$, for $t = 1, 2, \ldots, 5$, which are shown in solid lines in Figures 5.2(a)-(e), respectively.

Figure 5.2: Example to illustrate Procedure 5.1.



(a) T_1 (b) T_2 (c) T_3 (d) T_4 (e) T_5

It can be seen that T_1 is connected, and the subgraph of T_1 induced by $V(T^{(3)})$ is Φ, which equals L, and that the subgraph of T_1 induced by $V(T^{(6)})$ is $(\{6\}, \emptyset)$, which equals R. Thus, the subtree T_1 of $T^{(4)}$ is added to $\mathcal{Q}$. Similarly, it can be see that T_4 is also added to $\mathcal{Q}$. Since T_2 and T_3 are not connected, they are not added to $\mathcal{Q}$. Although T_5 is connected, the subgraph of T_5 induced by $V(T^{(3)})$ is $(\{3\}, \emptyset)$, which does not equal L. Thus, T_5 is not added to $\mathcal{Q}$. Therefore, we obtain that Procedure 5.1 returns $\mathcal{Q} = \{T_1, T_4\}$. □

We next prove Lemma 5.1 as follows to present, for each t where $1 \leq t \leq 5$, a sufficient and necessary condition for T_t to be added to $\mathcal{Q}$ in Step 2 of Procedure 5.1.

Lemma 5.1. *Consider Step 2 of Procedure 5.1. Then,*

(i) T_1 is added to $\mathcal{Q}$ if, and only if, $\beta(L) \vee \beta(R) = 1$;

(ii) T_2 is added to $\mathcal{Q}$ if, and only if, $\beta(L) = \beta(R) = 1$;

(iii) T_3 *is added to* $\mathcal{Q}$ *if, and only if,* $\alpha^{(l)}(L) = \beta(R) = 1$*;*

(iv) T_4 *is added to* $\mathcal{Q}$ *if, and only if,* $\beta(L) = \alpha^{(r)}(R) = 1$*;*

(v) T_5 *is added to* $\mathcal{Q}$ *if, and only if,* $\alpha^{(l)}(L) = \alpha^{(r)}(R) = 1$.

Proof. Firstly, if T_1 is added to $\mathcal{Q}$, then T_1 is connected. Thus, since $T_1 = (L \cup R) \cup G_1$ and G_1 is an empty tree, we know that either L or R must be empty, which implies that $\beta(L) \vee \beta(R) = 1$. Moreover, if $\beta(L) \vee \beta(R) = 1$, then the conditions for Step 2.b of Procedure 5.1 to add T_1 to $\mathcal{Q}$ are satisfied. Thus, (i) is proved. Secondly, if T_2 is added to $\mathcal{Q}$, then T_2 is connected. Thus, since $T_2 = (L \cup R) \cup G_2$ and G_2 contains v only, we know that both L and R must be empty, which implies that $\beta(L) = \beta(R) = 1$. Moreover, if $\beta(L) = \beta(R) = 1$, then the conditions for Step 2.b of Procedure 5.1 to add T_2 to $\mathcal{Q}$ are satisfied. Thus, (ii) is proved. Thirdly, if T_3 is added to $\mathcal{Q}$, then T_3 is connected. Thus, since $T_3 = (L \cup R) \cup G_3$ and G_3 contains (v, l) only, we know that R must be empty and L contains l, which implies that $\alpha^{(l)}(L) = \beta(R) = 1$. Moreover, if $\alpha^{(l)}(L) = \beta(R) = 1$, then the conditions for Step 2.b of Procedure 5.1 to add T_3 to $\mathcal{Q}$ are satisfied. Thus, (iii) is proved. Fourthly, the proof of (iv) is similar to the proof of (iii). Finally, if T_5 is added to $\mathcal{Q}$, then T_5 is connected. Thus, since $T_5 = (L \cup R) \cup G_5$ and G_5 contains only (v, l) and (v, r), we know that L must contain l and R must contain r, which implies that $\alpha^{(l)}(L) = \alpha^{(r)}(R) = 1$. Moreover, if $\alpha^{(l)}(L) = \alpha^{(r)}(R) = 1$, then the conditions for Step 2.b of Procedure 5.1 to add T_5 to $\mathcal{Q}$ are satisfied. Thus, (v) is proved. □

We can then establish Lemma 5.2 to present, for each t where $1 \leq t \leq 5$, the state of T_t as a subtree of $T^{(v)}$, if T_t is added to $\mathcal{Q}$ in Step 2 of Procedure 5.1.

Lemma 5.2. *Consider Step 2 of Procedure 5.1. Then,*

(i) If T_1 *is added to* $\mathcal{Q}$*, then* $w(T_1) = w(L) + w(R)$*,* $\alpha^{(v)}(T_1) = 0$*, and* $\beta(T_1) = \beta(L) \wedge \beta(R)$*;*

(ii) If T_2 is added to $\mathcal{Q}$, then $w(T_2) = w(L) + w(R)$, $\alpha^{(v)}(T_2) = 1$, and $\beta(T_2) = 0$;

(iii) If T_3 is added to $\mathcal{Q}$, then $w(T_3) = w(L) + w(R) + w(v, l)$, $\alpha^{(v)}(T_3) = 1$, and $\beta(T_3) = 0$;

(iv) If T_4 is added to $\mathcal{Q}$, then $w(T_4) = w(L) + w(R) + w(v, r)$, $\alpha^{(v)}(T_4) = 1$, and $\beta(T_4) = 0$;

(v) If T_5 is added to $\mathcal{Q}$, then $w(T_5) = w(L) + w(R) + w(v, l) + w(v, r)$, $\alpha^{(v)}(T_5) = 1$, and $\beta(T_5) = 0$.

Proof. For $1 \leq t \leq 5$, since $T_t = (L \cup R) \cup G_t$, the values of $w(T_t)$ can be easily verified by the definition of G_t. Moreover, since G_1 is an empty tree, and all G_t for $2 \leq t \leq 5$ contain v, we have that $\alpha^{(v)}(T_1) = 0$ and $\alpha^{(v)}(T_t) = 1$ for $2 \leq t \leq 5$. Finally, since all G_t for $2 \leq t \leq 5$ are not empty, we obtain that $\beta(T_t) = 0$ for $2 \leq t \leq 5$. Since $T_1 = (L \cup R) \cup G_1$ and G_1 is an empty tree, we obtain that T_1 is empty if, and only if, both L and R are empty, which implies that $\beta(T_1) = \beta(L) \wedge \beta(R)$. □

According to Lemma 5.2, we can determine the state of each tree in $\mathcal{Q}$, returned by Procedure 5.1, from the states of the given subtrees L and R of $T^{(l)}$ and $T^{(r)}$. Let us continue Example 5.2, for which $\mathcal{Q} = \{T_1, T_4\}$. The state of T_1, which is a subtree of $T^{(4)}$ shown in Figure 5.2(a), can be determined by (i) of Lemma 5.2 as follows. Since $w(L) = w(R) = 0$, $\beta(L) = 1$ and $\beta(R) = 0$, then $w(T_1) = w(L) + w(R) = 0$, $\alpha^{(4)}(T_1) = 0$, and $\beta(T_1) = \beta(L) \wedge \beta(R) = 0$, which implies that the state of T_1 is $(0, 0, 0)$. Similarly, according to (iv) of Lemma 5.2, we can obtain that the state of T_4, which is a subtree of $T^{(4)}$ shown in Figure 5.2(d), is $(0, 1, 0)$.

Moreover, for any vertex $v \in V$, where l and r are the left and the right children of v, consider any two subtrees L and L' of $T^{(l)}$ with $\alpha^{(l)}(L) = \alpha^{(l)}(L')$ and $\beta(L) = \beta(L')$, and two subtrees R and R' of $T^{(r)}$ with $\alpha^{(r)}(R) = \alpha_{(r)}(R')$ and $\beta(R) = \beta(R')$. For each t with $1 \leq t \leq 5$, Lemma 5.1 implies that $(L \cup R) \cup G_t$ is in the output returned

by applying Procedure 5.1 to L and R if, and only if, $(L' \cup R') \cup G_t$ is in the output returned by applying Procedure 5.1 to L' and R'. Moreover, by Lemma 5.2, we have that $\alpha^{(v)}((L' \cup R') \cup G_t) = \alpha^{(v)}((L \cup R) \cup G_t)$ and $\beta((L' \cup R') \cup G_t) = \beta((L \cup R) \cup G_t)$, and that if $w(L') \leq w(L)$ and $w(R') \leq w(R)$, then $w((L' \cup R') \cup G_t) \leq w((L \cup R) \cup G_t)$.

We can then prove the correctness of Procedure 5.1 in Lemma 5.3.

Lemma 5.3. *Procedure 5.1 runs in $O(1)$ time, and returns a set $\mathcal{Q}$, such that:*

(i) Each $Q \in \mathcal{Q}$ is a subtree of $T^{(v)}$, such that the subtree of Q induced by $V(T^{(l)})$ is equal to L, and that the subtree of Q induced by $V(T^{(r)})$ is equal to R.

(ii) Each Q of $T^{(v)}$, such that the subtree of Q induced by $V(T^{(l)})$ is equal to L, and that the subtree of Q induced by $V(T^{(r)})$ is equal to R, is in $\mathcal{Q}$.

Proof. It is easy to see that Procedure 5.1 runs in $O(1)$ time, and it is easy to verify from Step 2 that (i) holds for each $Q \in \mathcal{Q}$. To prove (ii), consider each subtree Q of $T^{(v)}$, such that the subtree of Q induced by $V(T^{(l)})$ is equal to L, and that the subtree of Q induced by $V(T^{(r)})$ is equal to R. Define $V' = V(Q) \setminus V(L) \setminus V(R)$ and $E' = E(Q) \setminus E(L) \setminus E(R)$. By the definition of Q, we have $V' \subseteq \{v\}$ and $E' \subseteq \{(v, l), (v, r)\}$. Consider the five cases for V' and E' as follows, where G_t for $1 \leq t \leq 5$ are defined in Step 1 of Procedure 5.1. Case 1: If $V' = \emptyset$, then $E' = \emptyset$, which implies that $Q = (L \cup R) \cup G_1$. Thus $Q \in \mathcal{Q}$. Case 2: If $V' = \{v\}$ and $E' = \emptyset$, then $Q = (L \cup R) \cup G_2$, which implies that $Q \in \mathcal{Q}$. Case 3: If $V' = \{v\}$ and $E' = \{(v, l)\}$, then $Q = (L \cup R) \cup G_3$, which implies that $Q \in \mathcal{Q}$. Case 4: If $V' = \{v\}$ and $E' = \{(v, r)\}$, then $Q = (L \cup R) \cup G_4$, which implies that $Q \in \mathcal{Q}$. Case 5: If $V' = \{v\}$ and $E' = \{(v, l), (v, r)\}$, then $Q = (L \cup R) \cup G_5$, which implies that $Q \in \mathcal{Q}$. Hence, Lemma 5.3 is proved. □

Based on Procedure 5.1, we can construct $\mathcal{S}^{(v)}$ for $v = 1, 2, ..., n$, iteratively, to enumerate all the tuples in $\mathcal{S}^{(n)} \times \mathcal{S}^{(n)}$ so as to find an optimal solution to the min-max 2-TSPT, which, however, has a running time exponential to n. Therefore, we

develop a dynamic programming as follows in Algorithm 5.2, to construct only a subset $\mathcal{H}^{(v)} \subseteq \mathcal{S}^{(v)} \times \mathcal{S}^{(v)}$, for $v = 1, 2, ..., n$, iteratively, such that the size of $\mathcal{H}^{(v)}$ for each v is bounded by a polynomial in U, where U is an upper bound on the optimal objective value to the min-max 2-TSPT. We will later show that this dynamic programming returns an optimal solution to any given instance of the min-max 2-TSPT.

During each iteration of Step 1 for $v = 1, 2, ..., n$, Algorithm 5.2 first constructs a subset $\mathcal{H}_1^{(v)}$ of $\mathcal{S}^{(v)} \times \mathcal{S}^{(v)}$ in Step 1.a as follows. Initially, $\mathcal{H}_1^{(v)}$ is set to be empty. If v is a leaf, then we have $T^{(v)} = (\{v\}, \emptyset)$, which has only two subtrees, $T^{(v)}$ and Φ. Thus, we set $\mathcal{H}_1^{(v)} = \{\Phi, T^{(v)}\} \times \{\Phi, T^{(v)}\}$. Otherwise, v has a left child and a right child, denoted by l and r respectively. Since $\max\{l, r\} \leq v - 1$, both $\mathcal{H}^{(l)}$ and $\mathcal{H}^{(r)}$ have been constructed. Thus, for each tuple $(L_1, L_2) \in \mathcal{H}^{(l)}$ and each tuple $(R_1, R_2) \in \mathcal{H}^{(r)}$, we apply Procedure 5.1 to join L_j and R_j to obtain a set $\mathcal{Q}_j$ of subtrees of $T^{(v)}$ in Step 1.a.i for $j = 1, 2$, and then we add all the tuples in $\mathcal{Q}_1 \times \mathcal{Q}_2$ to $\mathcal{H}_1^{(v)}$ in Step 1.a.ii.

Next, for each tuple $(Q_1, Q_2) \in \mathcal{H}_1^{(v)}$ obtained in Step 1.a, Algorithm 5.2 examines in Step 1.b whether or not $v \in V(Q_1) \cup V(Q_2)$, and whether or not $v \in V(Q_1) \cap V(Q_2)$ if $v = n$. Only the tuples that satisfy these conditions are kept to form a set $\mathcal{H}_2^{(v)}$ in Step 1.b.

Then, we construct $\mathcal{H}^{(v)}$ from $\mathcal{H}_2^{(v)}$ in Step 1.c and Step 1.d as follows. For each $w_1 \in \{0, 1..., U\}$, and for each $\alpha_j, \beta_j \in \{0, 1\}$ for $j = 1, 2$, we use $w_2^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2)$ to denote the smallest value of $w_2 \in \{0, 1, ..., U\}$, such that there exists at least one tuple $(Q_1, Q_2) \in \mathcal{H}_2^{(v)}$ with the state of Q_j equal to (w_j, α_j, β_j) for $j = 1, 2$. If there exists at least one tuple (Q_1', Q_2') such that the state of Q_1' is equal to (w_1, α_1, β_1) and the state of Q_2' is equal to $(w_2^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2), \alpha_2, \beta_2)$, we use $\mathcal{Q}^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2)$ to denote the set that consists any one of such tuples; otherwise, let $\mathcal{Q}^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2)$ be an empty set. Then, to construct $\mathcal{H}^{(v)}$, we only need to add all tuples in $\mathcal{Q}^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2)$ to $\mathcal{H}^{(v)}$, for each $w_1 \in \{0, 1..., U\}$,

and for each $\alpha_j, \beta_j \in \{0,1\}$ for $j = 1, 2$.

Finally, among all the tuples (Q_1, Q_2) stored in $\mathcal{H}^{(n)}$, Algorithm 5.2 identifies and returns a tuple (Q_1^*, Q_2^*) in Step 2 that minimizes $\max\{w(Q_1), w(Q_2)\}$.

Algorithm 5.2.

Input: *A tree $T = (V, E)$, an upper bound U on the optimal objective value, and an edge weight function w.*

Output: *A tuple (Q_1^*, Q_2^*) where Q_j^* is a subtree of T for $j = 1, 2$.*

1. For $v = 1, 2, ..., n$, do the following steps:

   (a) Set $\mathcal{H}_1^{(v)}$ to be empty. If v is a leaf, then set $\mathcal{H}_1^{(v)} = \{\Phi, T^{(v)}\} \times \{\Phi, T^{(v)}\}$. Otherwise, v has a left child and a right child, denoted by l and r respectively. Then, for each tuple $(L_1, L_2) \in \mathcal{H}^{(l)}$ and each tuple $(R_1, R_2) \in \mathcal{H}^{(r)}$, do the following steps:

      i. For $j = 1, 2$, apply Procedure 5.1 to join L_j and R_j to obtain a set $\mathcal{Q}_j$ of subtrees of $T^{(v)}$.

      ii. Add all the tuples in $\mathcal{Q}_1 \times \mathcal{Q}_2$ to $\mathcal{H}_1^{(v)}$.

   (b) Set $\mathcal{H}_2^{(v)}$ to be empty. For each tuple $(Q_1, Q_2) \in \mathcal{H}_1^{(v)}$, add (Q_1, Q_2) to $\mathcal{H}_2^{(v)}$ if both the following two conditions are satisfied:

      (i) $v \in V(Q_1) \cup V(Q_2)$, and

      (ii) if $v = n$, then $v \in V(Q_1) \cap V(Q_2)$.

   (c) For each $w_1 \in \{0, 1..., U\}$, and for each $\alpha_j, \beta_j \in \{0,1\}$ for $j = 1, 2$, set $w_2^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2) = +\infty$ and $\mathcal{Q}^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2) = \emptyset$. For each (Q_1, Q_2) in $\mathcal{H}_2^{(v)}$, do the following steps:

      i. Let $w(Q_1) = w_1$, $\alpha^{(v)}(Q_1) = \alpha_1$, $\alpha^{(v)}(Q_2) = \alpha_2$, $\beta(Q_1) = \beta_1$, $\beta(Q_2) = \beta_2$.

      ii. If $w(Q_2) < w_2^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2)$, set $w_2^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2) = w(Q_2)$, and set $\mathcal{Q}^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2) = \{(Q_1, Q_2)\}$.

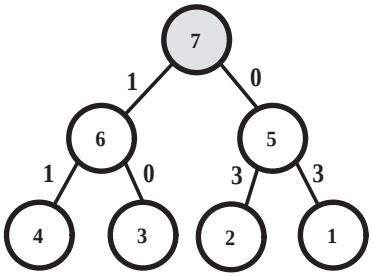
(d) Set $\mathcal{H}^{(v)}$ to be empty. For each $w_1 \in \{0, 1..., U\}$, and for each $\alpha_j, \beta_j \in \{0, 1\}$ for $j = 1, 2$, set $\mathcal{H}^{(v)} = \mathcal{H}^{(v)} \bigcup \mathcal{Q}^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2)$.

2. Among all the tuples (Q_1, Q_2) in $\mathcal{H}^{(n)}$, return (Q_1^*, Q_2^*) such that $\max\{w(Q_1), w(Q_2)\}$ is minimized. □

We use Example 5.3 to illustrate Algorithm 5.2.

Example 5.3. Consider the instance $[T, w, 2]$ shown in Figure 5.3, where vertices in T are labeled in a non-increasing order on their depths in T.

Figure 5.3: An instance $[T, w, 2]$ to illustrate Algorithm 5.2.



Let us apply Algorithm 5.2 on this instance, and explain the construction of $\mathcal{H}^{(v)}$ for $v = 6$ as follows, this being based on the construction of $\mathcal{H}^{(3)}$ and $\mathcal{H}^{(4)}$. Consider the construction of $\mathcal{H}^{(3)}$ in the iterations of Step 1 of Algorithm 5.2 for $v = 3$. It can be seen that Step 1.a adds $\{\Phi, T^{(3)}\} \times \{\Phi, T^{(3)}\}$ to $\mathcal{H}_1^{(3)}$. Then, as shown in Table 5.1, Step 1.b adds only $(\Phi, T^{(3)})$, $(T^{(3)}, \Phi)$, and $(T^{(3)}, T^{(3)})$ to $\mathcal{H}_2^{(3)}$, because vertex 3 is not in $V(\Phi) \cup V(\Phi)$. Moreover, by definition, we know that the state of Φ is $(0, 0, 1)$, and the state of $T^{(3)}$ is $(0, 1, 0)$. Thus, each subtree Q of $T^{(3)}$ satisfies $w(Q) = 0$ and $(\alpha^{(3)}(Q), \beta(Q)) \in \{(1, 0), (0, 1)\}$. Therefore, Step 1.c needs to consider only $w_1 = 0$ and $(\alpha_j, \beta_j) \in \{(1, 0), (0, 1)\}$ for $j = 1, 2$, and Step 1.d constructs $\mathcal{H}^{(3)}$ from $\mathcal{H}_2^{(3)}$, so that $\mathcal{H}^{(3)} = \{(\Phi, T^{(3)}), (T^{(3)}, \Phi), (T^{(3)}, T^{(3)})\}$, as shown in Table 5.2. For example, when $w_1 = 0$, $(\alpha_1, \beta_1) = (0, 1)$ and $(\alpha_2, \beta_2) = (1, 0)$, there exists one and only one tuple $(\Phi, T^{(3)}) \in \mathcal{H}_2^{(3)}$ such that the state of Φ is equal to (w_1, α_1, β_1), and that $(\alpha^{(3)}(T^{(3)}), \beta(T^{(3)})) = (\alpha_2, \beta_2)$. According to the definition of $w_2^{\min}(0, 0, 1, 1, 0)$, we

obtain that $w_2^{\min}(0,0,1,1,0) = 0$, and $\mathcal{Q}^{\min}(1,0,1,1,0) = \{(\Phi, T^{(3)})\}$. Similarly, for $v \in \{1,2,4\}$, we can obtain from Step 1 that $\mathcal{H}^{(v)} = \{(\Phi, T^{(v)}), (T^{(v)}, \Phi), (T^{(v)}, T^{(v)})\}$.

Table 5.1: Illustration of construction of $\mathcal{H}_2^{(3)}$ for the instance in Figure 5.3.

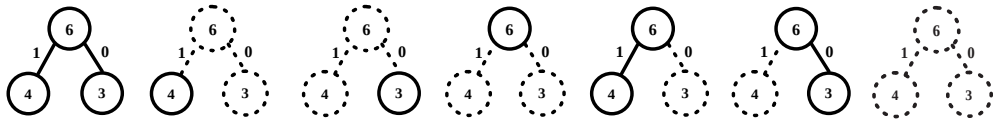
$\mathcal{H}_1^{(3)}$		$\mathcal{H}_2^{(3)}$
$(T^{(3)}, \Phi)$	$3 \in V(T^{(3)}) \bigcup V(\Phi)$	$(T^{(3)}, \Phi)$
$(\Phi, T^{(3)})$	$3 \in V(\Phi) \bigcup V(T^{(3)})$	$(\Phi, T^{(3)})$
$(T^{(3)}, T^{(3)})$	$3 \in V(T^{(3)}) \bigcup V(T^{(3)})$	$(T^{(3)}, T^{(3)})$
(Φ, Φ)	$3 \notin V(\Phi) \bigcup V(\Phi)$	

Table 5.2: Illustration of construction of $\mathcal{H}^{(3)}$ for the instance in Figure 5.3.

w_1	α_1	β_1	α_2	β_2	$\mathcal{H}_2^{(3)}$	$w_2^{\min}(w_1, \alpha_1, \alpha_2, \beta_1, \beta_2)$	$\mathcal{H}^{(3)}$
0	1	0	0	1	$(T^{(3)}, \Phi)$	0	$(T^{(3)}, \Phi)$
0	1	0	1	0	$(T^{(3)}, T^{(3)})$	0	$(T^{(3)}, T^{(3)})$
0	0	1	1	0	$(\Phi, T^{(3)})$	0	$(\Phi, T^{(3)})$
0	0	1	0	1		$+\infty$	$\emptyset$

Next, consider the iterations in Step 1 for $v = 6$, which has vertices 4 and 3 as the left and the right children, respectively. Let H_i for $i = 1, 2, ..., 7$ denote all the seven subtrees of $T^{(6)}$, these being shown in solid lines in Figures 5.4(a)-(g).

Figure 5.4: Illustration of each subtree H_i of $T^{(6)}$ for $i = 1, 2, \ldots, 7$.



(a) H_1 (b) H_2 (c) H_3 (d) H_4 (e) H_5 (f) H_6 (g) H_7

For each tuple $(L_1, L_2) \in \mathcal{H}^{(4)}$, and for each tuple $(R_1, R_2) \in \mathcal{H}^{(3)}$, Step 1.a.i of Algorithm 5.2 applies Procedure 5.1 to join L_j and R_j for $j = 1, 2$ to obtain a set $\mathcal{Q}_j$ of subtrees of $T^{(6)}$. We take $(L_1, L_2) = (\Phi, T^{(4)})$ and $(R_1, R_2) = (\Phi, T^{(3)})$ as an

example to illustrate this step as follows. According to Procedure 5.1, $\mathcal{Q}_1$ contains H_4 and H_7, which is returned by joining Φ and Φ, and $\mathcal{Q}_2$ contains H_1, which is returned by joining $T^{(3)}$ and $T^{(4)}$. Hence, Step 1.a.i of Algorithm 5.2 adds $\{H_4, H_7\} \times \{H_1\}$ to $\mathcal{H}_1^{(6)}$. Furthermore, since (H_4, H_1) and (H_7, H_1) both satisfy conditions (i) and (ii) in Step 1.b, they are further added to $\mathcal{H}_2^{(6)}$. Similarly, by enumerating each tuple $(L_1, L_2) \in \mathcal{H}^{(4)}$ and each tuple $(R_1, R_2) \in \mathcal{H}^{(3)}$, we can obtain all the tuples in $\mathcal{H}_1^{(6)}$ and $\mathcal{H}_2^{(6)}$, as shown in Table 5.3.

Table 5.3: Construction of $\mathcal{H}_1^{(6)}$ and $\mathcal{H}_2^{(6)}$ for the instance in Figure 5.3.

(L_1, L_2):(R_1, R_2)	$\mathcal{H}_1^{(6)}$	$\mathcal{H}_2^{(6)}$
$(T^{(4)}, \Phi)$:$(T^{(3)}, \Phi)$	(H_1, H_4),(H_1, H_7)	(H_1, H_4),(H_1, H_7)
$(\Phi, T^{(4)})$:$(T^{(3)}, \Phi)$	(H_3, H_2),(H_3, H_5), (H_6, H_2),(H_6, H_5)	(H_3, H_5),(H_6, H_2), (H_6, H_5)
$(T^{(4)}, T^{(4)})$:$(T^{(3)}, \Phi)$	(H_1, H_2),(H_1, H_5)	(H_1, H_2),(H_1, H_5)
$(T^{(4)}, \Phi)$:$(\Phi, T^{(3)})$	(H_2, H_6),(H_2, H_3), (H_5, H_3),(H_5, H_6)	(H_2, H_6),(H_5, H_3), (H_5, H_6)
$(\Phi, T^{(4)})$:$(\Phi, T^{(3)})$	(H_4, H_1),(H_7, H_1)	(H_4, H_1),(H_7, H_1)
$(T^{(4)}, T^{(4)})$:$(\Phi, T^{(3)})$	(H_2, H_1),(H_5, H_1)	(H_2, H_1),(H_5, H_1)
$(T^{(4)}, \Phi)$:$(T^{(3)}, T^{(3)})$	(H_1, H_3),(H_1, H_6)	(H_1, H_3),(H_1, H_6)
$(\Phi, T^{(4)})$:$(T^{(3)}, T^{(3)})$	(H_3, H_1),(H_6, H_1)	(H_3, H_1),(H_6, H_1)
$(T^{(4)}, T^{(4)})$:$(T^{(3)}, T^{(3)})$	(H_1, H_1)	(H_1, H_1)

According to Lemma 5.2, the state of H_i, which is denoted by S_i, for $i = 1, 2, \ldots, 7$ can be determined as follows:

$$S_1 = (1, 1, 0), \quad S_2 = (0, 0, 0), \quad S_3 = (0, 0, 0),$$
$$S_4 = (0, 1, 0), \quad S_5 = (1, 1, 0), \quad S_6 = (0, 1, 0), \quad S_7 = (0, 0, 1).$$

Thus, each subtree Q of $T^{(6)}$ satisfies that $w(Q) \in \{0, 1\}$, and that if $w(Q) = 1$, then $(\alpha^{(6)}(Q), \beta(Q)) = (1, 0)$, and otherwise, $(\alpha^{(6)}(Q), \beta(Q)) \in \{(0, 1), (1, 0), (0, 0)\}$. Therefore, in Step 1.c of Algorithm 5.2, if $w_1 = 1$, we need to consider only $(\alpha_1, \beta_1) =$

$(1,0)$ and $(\alpha_2,\beta_2) \in \{(0,1),(1,0),(0,0)\}$, and otherwise, $w_1 = 0$, and we need to consider only $(\alpha_i,\beta_i) \in \{(0,1),(1,0),(0,0)\}$ for $i = 1,2$. Accordingly, we can obtain $\mathcal{H}^{(6)}$ from $\mathcal{H}_2^{(6)}$ as shown in Table 5.4.

Table 5.4: Illustration of construction of $\mathcal{H}^{(6)}$ for the instance in Figure 5.3.

w_1	α_1	β_1	α_2	β_2	$\mathcal{H}_2^{(6)}$	$w_2^{\min}(w_1,\alpha_1,\alpha_2,\beta_1,\beta_2)$	$\mathcal{H}^{(6)}$
1	1	0	0	1	(H_1,H_7)	0	(H_1,H_7)
1	1	0	1	0	(H_1,H_1),(H_1,H_4),(H_1,H_5), (H_1,H_6),(H_5,H_1),(H_5,H_6)	0	(H_1,H_4)
1	1	0	0	0	(H_1,H_3),(H_1,H_2),(H_5,H_3)	0	(H_1,H_3)
0	0	1	0	1		$+\infty$	$\emptyset$
0	0	1	1	0	(H_7,H_1)	1	(H_7,H_1)
0	0	1	0	0		$+\infty$	$\emptyset$
0	1	0	0	1		$+\infty$	$\emptyset$
0	1	0	1	0	(H_4,H_1),(H_6,H_5),(H_6,H_1)	1	(H_6,H_1)
0	1	0	0	0	(H_6,H_2)	0	(H_6,H_2)
0	0	0	0	1		$+\infty$	$\emptyset$
0	0	0	1	0	(H_2,H_6),(H_2,H_1),(H_3,H_5) (H_3,H_1)	0	(H_2,H_6)
0	0	0	0	0		$+\infty$	$\emptyset$

For example, when $w_1 = 1$, $(\alpha_1,\beta_1) = (1,0)$ and $(\alpha_2,\beta_2) = (0,0)$, there exist three tuples $(Q_1,Q_2) \in \mathcal{H}_2^{(6)}$ such that the state of Q_1 is (w_1,α_1,β_1) and $(\alpha^{(6)}(Q_2),\beta(Q_2)) = (\alpha_2,\beta_2)$. These three tuples in $\mathcal{H}_2^{(6)}$ can be seen to be (H_1,H_3), (H_1,H_2) and (H_5,H_3). Since $w(H_3) = 0$ and $w(H_2) = 0$, then $w_2^{\min} = 0$ for the case $w_1 = 1$, $(\alpha_1,\beta_1) = (1,0)$ and $(\alpha_2,\beta_2) = (0,0)$, and $\mathcal{Q}^{\min}(1,1,0,0,0) = \{(H_1,H_3)\}$. □

To show the correctness of Algorithm 5.2, we first establish Lemma 5.4, which implies that each tuple $(Q_1,Q_2) \in \mathcal{H}^{(n)}$, obtained in Step 1 of Algorithm 5.2, covers all the vertices in T.

Lemma 5.4. *For each $v = 1,2,...,n$, and for each tuple $(Q_1,Q_2) \in \mathcal{H}^{(v)}$ obtained in Step 1 of Algorithm 5.2, it satisfies that $V(T^{(v)}) = V(Q_1) \cup V(Q_2)$.*

Proof. We prove Lemma 5.4 by induction. For each leaf v of T, it can be verified from Step 1 of Algorithm 5.2 that $V(T^{(v)}) = V(Q_1) \cup V(Q_2)$ for each $(Q_1, Q_2) \in \mathcal{H}^{(v)}$. Consider any vertex $v \in V$ that is not a leaf. Let l and r denote the left and right children of v, respectively. Suppose that $V(T^{(u)}) = V(Q'_1) \cup V(Q'_2)$ holds for each $(Q'_1, Q'_2) \in \mathcal{H}^{(u)}$, for $1 \leq u \leq v-1$. Consider each $(Q_1, Q_2) \in \mathcal{H}^{(v)}$. For $j = 1, 2$, let L_j and R_j denote the subtrees of Q_j induced by $V(T^{(l)})$ and $V(T^{(r)})$ respectively. Due to Step 1.a.i and Lemma 5.3, we have $V(Q_1) \cup V(Q_2) \subseteq V(T^{(v)})$, $(L_1, L_2) \in \mathcal{H}^{(l)}$, and $(R_1, R_2) \in \mathcal{H}^{(r)}$. Since $\max\{l, r\} \leq v-1$, we obtain that $V(T^{(l)}) = V(L_1) \cup V(L_2)$ and $V(T^{(r)}) = V(R_1) \cup V(R_2)$, which implies that $V(T^{(v)}) \setminus \{v\} \subseteq V(Q_1) \cup V(Q_2)$. Moreover, by Step 1.b, we know $v \in V(Q_1) \cup V(Q_2)$. Thus, we obtain $V(T^{(v)}) \subseteq V(Q_1) \cup V(Q_2)$, which, together with $V(Q_1) \cup V(Q_2) \subseteq V(T^{(v)})$, implies that $V(T^{(v)}) = V(Q_1) \cup V(Q_2)$. Lemma 5.4 is proved. □

Consider the optimal solution $(\widetilde{T}_1, \widetilde{T}_2)$ to any instance of the min-max 2-TSPT. We can then establish Lemma 5.5 to further prove the correctness of Algorithm 5.2.

Lemma 5.5. *For each $v = 1, 2, ...n$, there exists a tuple $(Q_1, Q_2) \in \mathcal{H}^{(v)}$, obtained in Step 1 of Algorithm 5.2, such that the state of Q_1 is the same as the state of $\widetilde{T}_1^{(v)}$, and that $w(Q_2) \leq w(\widetilde{T}_2^{(v)})$, $\alpha^{(v)}(Q_2) = \alpha^{(v)}(\widetilde{T}_2^{(v)})$, and $\beta(Q_2) = \beta(\widetilde{T}_2^{(v)})$, where $\widetilde{T}_i^{(v)}$ denotes the subtree of $\widetilde{T}_i$ induced by $V(T^{(v)})$, for $i = 1, 2$.*

Proof. We prove Lemma 5.5 by induction. For each leaf v of T, we know that $\widetilde{T}_j^{(v)} \in \{\Phi, T^{(v)}\}$ for $j = 1, 2$, which implies that $(\widetilde{T}_1^{(v)}, \widetilde{T}_2^{(v)}) \in \mathcal{H}_1^{(v)}$, according to Step 1.a of Algorithm 5.2. By $v \in V(\widetilde{T}_1) \cup V(\widetilde{T}_2)$ and $n \in V(\widetilde{T}_1) \cap V(\widetilde{T}_2)$, according to Step 1.b of Algorithm 5.2, we obtain $(\widetilde{T}_1^{(v)}, \widetilde{T}_2^{(v)}) \in \mathcal{H}_2^{(v)}$. Thus, from Step 1.c and Step 1.d of Algorithm 5.2, we obtain that Lemma 5.5 holds for the leaf v.

Next, consider any $v \in V$ that is not a leaf. Let l and r denote the left and right children of $T^{(v)}$, respectively. Assume that Lemma 5.5 holds for each u where $1 \leq u \leq v-1$. We are going to show as follows that Lemma 5.5 holds for v.

Since $\max\{l, r\} \leq v - 1$, we know that there exist $L_j \in \mathcal{H}^{(l)}$ and $R_j \in \mathcal{H}^{(r)}$ for $j = 1, 2$, such that the states of L_1 and R_1 are equal to the states of $\widetilde{T}_1^{(l)}$ and $\widetilde{T}_1^{(r)}$ respectively, that $w(L_2) \leq w(\widetilde{T}_2^{(l)})$, $\alpha^{(l)}(L_2) = \alpha^{(l)}(\widetilde{T}_2^{(l)})$, $\beta(L_2) = \beta(\widetilde{T}_2^{(l)})$, and that $w(R_2) \leq w(\widetilde{T}_2^{(r)})$, $\alpha^{(r)}(R_2) = \alpha^{(l)}(\widetilde{T}_2^{(r)})$, and $\beta(R_2) = \beta(\widetilde{T}_2^{(r)})$. Let $\mathcal{Q}_j$ denote the set returned by Procedure 5.1 in Step 1.a.i of Algorithm 5.2 to join L_j and R_j, for $j = 1, 2$.

For $j = 1, 2$, since $\widetilde{T}_j^{(v)}$ is a subtree of $T^{(v)}$, and since the subtrees of $\widetilde{T}_j^{(v)}$ induced by $V(T^{(l)})$ and $V(T^{(r)})$ are equal to $\widetilde{T}_j^{(l)}$ and $\widetilde{T}_j^{(r)}$ respectively, from Lemma 5.3, it can be seen that by applying Procedure 5.1 to join $\widetilde{T}_j^{(l)}$ and $\widetilde{T}_j^{(r)}$, we can obtain a set $\widetilde{\mathcal{Q}}_j$, such that $\widetilde{T}_j^{(v)} \in \widetilde{\mathcal{Q}}_j$, and that $\widetilde{T}_j^{(v)} = (\widetilde{T}_j^{(l)} \cup \widetilde{T}_j^{(r)}) \cup G_t$ for some t with $1 \leq t \leq 5$, where G_t is the same as that defined in Step 1 of Procedure 5.1. Let $Q'_j = (L_j \cup R_j) \cup G_t$. Since $\alpha^{(l)}(L_j) = \alpha^{(l)}(\widetilde{T}_j^{(l)})$, $\beta(L_j) = \beta(\widetilde{T}_j^{(l)})$, $\alpha^{(r)}(R_j) = \alpha^{(r)}(\widetilde{T}_j^{(r)})$, and $\beta(R_j) = \beta(\widetilde{T}_j^{(r)})$, from Lemma 5.1 and $(\widetilde{T}_j^{(l)} \cup \widetilde{T}_j^{(r)}) \cup G_t \in \widetilde{\mathcal{Q}}_j$ we have $Q'_j \in \mathcal{Q}_j$. Hence, due to Step 1.a.i and Step 1.a.ii of Algorithm 5.2, we obtain $(Q'_1, Q'_2) \in \mathcal{H}_1^{(v)}$.

Since the states of L_1 and R_1 are equal to the states of $\widetilde{T}_1^{(l)}$ and $\widetilde{T}_1^{(r)}$ respectively, due to Lemma 5.2, we have $\alpha^{(v)}(Q'_1) = \alpha^{(v)}(\widetilde{T}_1^{(v)})$, $\beta(Q'_1) = \beta(\widetilde{T}_1^{(v)})$, and $w(Q'_1) = w(\widetilde{T}_1^{(v)})$. Moreover, since $w(L_2) \leq w(\widetilde{T}_2^{(l)})$, $\alpha^{(l)}(L_2) = \alpha^{(l)}(\widetilde{T}_2^{(l)})$, $\beta(L_2) = \beta(\widetilde{T}_2^{(l)})$, $w(R_2) \leq w(\widetilde{T}_2^{(r)})$, $\alpha^{(r)}(R_2) = \alpha^{(l)}(\widetilde{T}_2^{(r)})$, and $\beta(R_2) = \beta(\widetilde{T}_2^{(r)})$, from Lemma 5.2, we obtain $\alpha^{(v)}(Q'_2) = \alpha^{(v)}(\widetilde{T}_2^{(v)})$, $\beta(Q'_2) = \beta(\widetilde{T}_2^{(v)})$, and $w(Q'_2) \leq w(\widetilde{T}_2^{(v)})$. Thus, since $\widetilde{T}_1^{(v)}$ and $\widetilde{T}_2^{(v)}$ satisfy the two conditions in Step 1.b of Algorithm 5.2, it is easy to see that Q'_1 and Q'_2 also satisfy the two conditions, which implies that $(Q'_1, Q'_2) \in \mathcal{H}_2^{(v)}$. Hence, since $w(Q'_1) = w(\widetilde{T}_1^{(v)})$ and $w(Q'_2) \leq w(\widetilde{T}_2^{(v)})$, noticing that $w(Q'_1) \leq U$ and $w(Q'_2) \leq U$, from Step 1.c and Step 1.d of Algorithm 5.2, we obtain that Lemma 5.5 holds for v, which completes the proof. □

From Lemma 5.4 and Lemma 5.5, we can establish Theorem 5.2, which shows the correctness and time complexity of Algorithm 5.2.

Theorem 5.2. *Algorithm 5.2 returns an optimal solution to any given instance of the min-max 2-TSPT in $O(nU^2)$ time.*

Proof. Due to Step 1 of Algorithm 5.2, $|\mathcal{H}^{(v)}|$ is $O(U)$ for all the leaves. Thus, in Step 1.a of Algorithm 5.2, if v is a not a leaf, it takes at most $O(U^2)$ time to construct $\mathcal{H}_1^{(v)}$ from $\mathcal{H}^{(l)}$ and $\mathcal{H}^{(r)}$. To construct $\mathcal{H}_2^{(v)}$ from $\mathcal{H}_1^{(v)}$, we only need to enumerate each tuple in $\mathcal{H}_1^{(v)}$ just once. Therefore, it takes $O(U^2)$ time to construct $\mathcal{H}_2^{(v)}$ from $\mathcal{H}_1^{(v)}$. According to Step 1.c, it needs to enumerate each tuple in $\mathcal{H}_2^{(v)}$ exactly once. Therefore, it takes $O(U^2)$ time to construct $\mathcal{H}^{(v)}$ from $\mathcal{H}_2^{(v)}$. In summary, it can be seen that the running time of Step 1 of Algorithm 5.2 is at most $O(U^2)$ for $v = 1, 2, \ldots, n$. Thus, the running time of Algorithm 5.2 is $O(nU^2)$.

Consider (Q_1^*, Q_2^*) returned by Algorithm 5.2, which implies that $(Q_1^*, Q_2^*) \in \mathcal{H}^{(n)}$. From Step 1.b of Algorithm 5.2, we know that both Q_1^* and Q_2^* contain the root n of the tree T. By Lemma 5.4, we know that $V = V(Q_1^*) \cup V(Q_2^*)$, which implies that (Q_1^*, Q_2^*) is a feasible solution. Moreover, consider the optimal solution $(\widetilde{T}_1, \widetilde{T}_2)$. By Lemma 5.5, we know that there exists a tuple $(Q_1, Q_2) \in \mathcal{H}^{(n)}$, such that $w(Q_1) = w(\widetilde{T}_1)$ and $w(Q_2) \leq w(\widetilde{T}_2)$, which implies that $\max\{w(Q_1), w(Q_2)\} \leq \max\{w(\widetilde{T}_1), w(\widetilde{T}_2)\}$. Thus, due to Step 2 of Algorithm 5.2, we have $\max\{w(Q_1^*), w(Q_2^*)\} \leq \max\{w(Q_1), w(Q_2)\}$, which implies that $\max\{w(Q_1^*), w(Q_2^*)\} \leq \max\{w(\widetilde{T}_1), w(\widetilde{T}_2)\}$. Since $(\widetilde{T}_1, \widetilde{T}_2)$ is an optimal solution, we obtain that $\max\{w(Q_1^*), w(Q_2^*)\} = \max\{w(\widetilde{T}_1), w(\widetilde{T}_2)\}$. Hence, (Q_1^*, Q_2^*) is also an optimal solution. □

Finally, we can establish Lemma 5.6 as follows, which implies that $w(T)/2 \leq \text{OPT} \leq w(T)$, where OPT is the objective value to an optimal solution to $[T, w, 2]$. Thus, we can choose $U = w(T)$ for Algorithm 5.2, so that the time complexity of Algorithm 5.2 is $O(nW^2)$, which is a pseudo-polynomial time.

Lemma 5.6. *For any instance $[T, w, 2]$ of the min-max 2-TSPT, the optimal objective value* OPT *to $[T, w, 2]$ satisfies $w(T)/2 \leq \text{OPT} \leq w(T)$.*

Proof. Consider the optimal solution $(\widetilde{T}_1, \widetilde{T}_2)$. Since $V(T) \subseteq V(\widetilde{T}_1) \cup V(\widetilde{T}_2)$ and $n \in V(\widetilde{T}_1) \cap V(\widetilde{T}_2)$, we obtain $w(T) \leq w(\widetilde{T}_1) + w(\widetilde{T}_2)$, which implies that $w(T)/2 \leq \max\{w(\widetilde{T}_1), w(\widetilde{T}_2)\} = \text{OPT}$. Moreover, since (T, Φ) is a feasible solution, we obtain $\text{OPT} \leq w(T)$. Thus, Lemma 5.6 is proved. □

5.3.3 An FPTAS for the min-max 2-TSPT

Based on Algorithm 5.2, which is a pseudo-polynomial time exact algorithm for the min-max 2-TSPT, we develop a fully polynomial time approximation scheme (FPTAS) as follows in Algorithm 5.3. For any given $\epsilon > 0$ and any instance $[T, w, 2]$, Algorithm 5.3 first scales the edge weight $w(e)$ for each $e \in E(T)$ to $w'(e) = \lfloor w(e)/\delta \rfloor$ in Step 1, where $\delta = \epsilon w(T)/(2|E(T)|)$, so as to create a new instance $[T, w', 2]$. It can be seen that $w'(T) \leq 2|E(T)|/\epsilon \leq 2(n-1)/\epsilon$. Thus, since $\{T, \Phi\}$ is a feasible solution to $[T, w', 2]$, the objective value of an optimal solution to the instance $[T, w', 2]$ is less than or equal to $2(n-1)/\epsilon$. Algorithm 5.3 then applies Algorithm 5.2 with $U = w'(T)$ to $[T, w', 2]$ in Step 2, to find an optimal solution (Q_1^*, Q_2^*) to $[T, w', 2]$, which is also a feasible solution to $[T, w, 2]$, and is returned in Step 3.

Algorithm 5.3.

Input: *Any constant $\epsilon > 0$, and an instance $[T, w, 2]$ of the min-max 2-TSPT.*

Output: *A feasible solution (Q_1^*, Q_2^*).*

1. let $\delta = \epsilon w(T)/(2|E(T)|)$. For each $e \in E(T)$, let $w'(e) = \lfloor w(e)/\delta \rfloor$ to construct a new instance $[T, w', 2]$.

2. Apply Algorithm 5.2 to obtain an optimal solution, denoted by (Q_1^*, Q_2^*), to the instance $[T, w', 2]$.

3. Return (Q_1^*, Q_2^*). □

We can establish Theorem 5.3 as follows, which implies that Algorithm 5.3 is an FPTAS for the min-max 2-TSPT.

Theorem 5.3. *Algorithm 5.3 is an FPTAS for the min-max 2-TSPT.*

Proof. Consider the optimal solution $(\widetilde{T}_1, \widetilde{T}_2)$ to the problem instance $[T, w, 2]$, which is also a feasible solution to the problem instance $[T, w', 2]$. Since (Q_1^*, Q_2^*) returned by Algorithm 5.3 is an optimal solution to $[T, w', 2]$, we obtain that (Q_1^*, Q_2^*) is also a feasible solution to $[T, w, 2]$, and that $\max\{w'(Q_1^*), w'(Q_2^*)\} \leq \max\{w'(\widetilde{T}_1), w'(\widetilde{T}_2)\}$. Moreover, for $j = 1, 2$, we have

$$w(Q_j^*) \leq \sum_{e \in E(Q_j^*)} (w'(e) + 1)\delta \leq \delta w'(Q_j^*) + \delta |E(Q_j^*)|,$$

which implies that

$$\begin{aligned} \max\{w(Q_1^*), w(Q_2^*)\} &\leq \delta \max\{w'(Q_1^*), w'(Q_2^*)\} + \delta |E(T)| \\ &\leq \delta \max\{w'(\widetilde{T}_1), w'(\widetilde{T}_2)\} + \epsilon w(T)/2. \end{aligned}$$

Thus, since $w'(\widetilde{T}_j) \leq w(\widetilde{T}_j)/\delta$ for $j = 1, 2$, we obtain that $\delta \max\{w'(\widetilde{T}_1), w'(\widetilde{T}_2)\} \leq$ OPT. By Lemma 5.6, $w(T)/2 \leq$ OPT. Thus, we obtain that $\max\{w(Q_1^*), w(Q_2^*)\} \leq (1 + \epsilon)$OPT.

Moreover, since T is a tree, we have $|E(T)| \leq n - 1$. Thus,

$$w'(T) = \sum_{e \in E(T)} w'(e) \leq \sum_{e \in E(T)} w(e)/\delta \leq w(T)/\delta = 2|E(T)|/\epsilon \leq 2(n-1)/\epsilon,$$

which, together with Theorem 5.2, implies that the running time of Algorithm 5.3 is in $O(n^3/\epsilon^2)$. Hence, Theorem 5.3 is proved. □

5.4 Extensions

In this section, we first generalize the dynamic programming approach to develop a pseudo-polynomial time exact algorithm and an FPTAS for the min-max k-TSPT in

Section 5.4.1, for any given constant $k \geq 2$. We then further generalize the algorithms for two other variants of the min-max k-TSPT in Section 5.4.2 and Section 5.4.3, where multiple depots and demand points are taken into consideration, respectively. We then investigate a min-sum k-TSPT, with a min-sum objective to minimize the total weight of routes of the vehicles. For any given constant $k \geq 2$, by generalizing the dynamic programming approach derived for the min-max k-TSPT, we obtain the first polynomial time exact algorithm for the min-sum k-TSPT with multiple depots in Section 5.4.4.

5.4.1 Min-max k-TSPT

The min-max k-TSPT is an extension of the min-max 2-TSPT by taking into consideration k vehicles with k a given constant independent of the input size greater than one, which aims to determine a k-subtree tuple $(Q_1, Q_2, ..., Q_k)$ of T such that the depot is contained in every subtree of $(Q_1, Q_2, ..., Q_k)$ and the union of $V(Q_j)$ for $1 \leq j \leq k$ covers V, with the the maximum total edge weight of the k subtrees minimized. We use $[T, w, k]$ to denote an instance of the k-TSPT, and adopt the same definition as Definition 5.1 to define standard instances. Thus, Algorithm 5.1 can transform any instance $[T, w, k]$ of the min-max k-TSPT to a standard instance $[T', w', k]$. By following arguments similarly to those in the proof of Theorem 5.1, we can obtain that the resulting standard instance $[T', w', k]$ is equivalent to $[T, w, k]$. Thus, without loss of generality, we assume that any given instance $[T, w, k]$ of the min-max k-TSPT is standard, and that vertices in $V = \{1, 2, ..., n\}$ are labeled in a non-increasing order on their depths in T, so that n is the root and the depot of T.

By extending the dynamic programming for the min-max 2-TSPT in Algorithm 5.2, we can obtain a dynamic programming for the k-TSPT, as shown in Algorithm 5.4. Algorithm 5.4 constructs $\mathcal{H}^{(v)}$ for $v = 1, 2, ..., n$, iteratively, where $\mathcal{H}^{(v)}$ here is a set of k-subtree tuples $(Q_1, Q_2, ..., Q_k)$ with Q_j being a subtree of $T^{(v)}$ for $1 \leq j \leq k$. Let

$\tilde{w} = (w_1, w_2, \ldots, w_{k-1})$, $\tilde{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_k)$, and $\tilde{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$. Similar to the min-max 2-TSPT, for each $\tilde{w} \in \{0, 1..., U\}^{k-1}$, each $\tilde{\alpha} \in \{0,1\}^k$, and each $\tilde{\beta} \in \{0,1\}^k$, we use $w_k^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta})$ to denote the smallest value of $w_k \in \{0, 1, ..., U\}$, such that there exists at least one tuple $(Q_1, Q_2, \ldots, Q_k) \in \mathcal{H}_2^{(v)}$ with the state of Q_j equal to (w_j, α_j, β_j) for $j = 1, 2, \ldots, k$. If there exists at least one tuple $(Q'_1, Q'_2, \ldots, Q'_k)$ such that the state of Q'_j is equal to (w_j, α_j, β_j) for $j = 1, 2, \ldots, k-1$, and the state of Q'_k is equal to $(w_k^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}), \alpha_k, \beta_k)$, we use $\mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta})$ to denote the set that consists any one of such tuples; otherwise, let $\mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta})$ to be an empty set.

Algorithm 5.4.

Input: *A tree $T = (V, E)$, an upper bound U on the optimal objective value, and an integer $k \geq 2$.*

Output: *A k-subtree tuple $(Q_1^*, Q_2^*, .., Q_k^*)$ where Q_j^* is a subtree of T for $j = 1, 2, ..., k$.*

1. For $v = 1, 2, ..., n$, do the following steps:

   (a) Set $\mathcal{H}_1^{(v)}$ to be empty. If v is a leaf, then let $\mathcal{H}_1^{(v)}$ be the set of all tuples $(Q_1, Q_2, ..., Q_k)$ with $Q_j \in \{\Phi, T^{(v)}\}$ for $1 \leq j \leq k$. Otherwise, v has a left child and a right child, denoted by l and r respectively. Then, for each tuple $(L_1, L_2, ..., L_k) \in \mathcal{H}^{(l)}$ and each tuple $(R_1, R_2, ..., R_k) \in \mathcal{H}^{(r)}$, do the following steps:

      i. For $1 \leq j \leq k$, apply Procedure 5.1 to join L_j and R_j to obtain a set $\mathcal{Q}_j$ of subtrees of $T^{(v)}$.

      ii. Add all the tuples in $\mathcal{Q}_1 \times \mathcal{Q}_2 \times ... \times \mathcal{Q}_k$ to $\mathcal{H}_1^{(v)}$.

   (b) Set $\mathcal{H}_2^{(v)}$ to be empty. For each tuple $(Q_1, Q_2, ..., Q_k) \in \mathcal{H}_1^{(v)}$, add $(Q_1, Q_2..., Q_k)$ to $\mathcal{H}_2^{(v)}$ if both the following two conditions are satisfied:

      (i) $v \in \bigcup_{j=1}^k V(Q_j)$, and

      (ii) if $v = n$, then $v \in \bigcap_{j=1}^k V(Q_j)$.

(c) For each $\tilde{w} \in \{0, 1..., U\}^{k-1}$, each $\tilde{\alpha} \in \{0,1\}^k$, and each $\tilde{\beta} \in \{0,1\}^k$, set $w_k^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}) = +\infty$ and $\mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}) = \emptyset$. For each $(Q_1, Q_2, \ldots, Q_k)$ in $\mathcal{H}_2^{(v)}$, do the following steps:

i. Let $(w(Q_1), w(Q_2), \ldots, w(Q_{k-1})) = \tilde{w}$, $(\alpha^{(v)}(Q_1), \alpha^{(v)}(Q_2), \ldots, \alpha^{(v)}(Q_k)) = \tilde{\alpha}$, $(\beta(Q_1), \beta(Q_2), \ldots, \beta(Q_k)) = \tilde{\beta}$.

ii. If $w(Q_k) < w_k^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta})$, set $w_2^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}) = w(Q_k)$, and set $\mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}) = \{(Q_1, Q_2, \ldots, Q_k)\}$.

(d) Set $\mathcal{H}^{(v)}$ to be empty. For each $\tilde{w} \in \{0, 1..., U\}^{k-1}$, each $\tilde{\alpha} \in \{0,1\}^k$, and each $\tilde{\beta} \in \{0,1\}^k$, let $\mathcal{H}^{(v)} = \mathcal{H}^{(v)} \bigcup \mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta})$.

2. Identify the tuple $(Q_1^*, Q_2^*, ..., Q_k^*)$ among all the tuples $(Q_1, Q_2, ..., Q_k)$ in $\mathcal{H}^{(n)}$, such that $\max\{w(Q_1), w(Q_2), ..., w(Q_k)\}$ is minimized. Return $(Q_1^*, Q_2^*, ..., Q_k^*)$.

□

Similar to the proofs of Lemma 5.4, Lemma 5.5, and Theorem 5.2, it is able to show that Algorithm 5.4 returns an optimal solution to any given instance of the min-max k-TSPT in $O(nU^{2(k-1)})$ time. Moreover, similar to the proof of Lemma 5.6, it is able to show that $w(T)/k \leq \text{OPT} \leq w(T)$, where OPT is the optimal objective value to $[T, w, k]$. Thus, we can choose $U = w(T)$ for Algorithm 5.4, so that the running time of Algorithm 5.4 is $O(nW^{2(k-1)})$, which is pseudo-polynomial when k is a constant independent of the input size.

Based on Algorithm 5.3, for any given $\epsilon > 0$, we can then develop a $(1+\epsilon)$-approximation scheme for the min-max k-TSPT. Given any instance $[T, w, k]$ of the min-max k-TSPT and any $\epsilon > 0$, we first scale $w(e)$ to $w'(e)$ for each $e \in E(T)$ by following Step 1 of Algorithm 5.3, so as to obtain a scaled instance $[T, w', k]$ of the min-max k-TSPT. We then apply Algorithm 5.4, instead of Algorithm 5.2, on $[T, w', k]$ with $U = w'(T)$ to obtain an optimal solution to the scaled instance $[T, w', k]$, which is denoted by $(Q_1^*, Q_2^*..., Q_k^*)$. Similar to the proof of Theorem 5.3, it is able to

show that $\max\{w(Q_1^*), w(Q_2^*), ..., w(Q_k^*)\} \leq (1+\epsilon)\text{OPT}$, which implies that the above algorithm is an $(1+\epsilon)$-approximation scheme for the min-max k-TSPT. Moreover, since the approximation scheme has a running time in $O(n^{2k-1}/\epsilon^{2k-2})$, it is an FPTAS if k is a given constant independent of the input size.

Finally, we show as follows in Theorem 5.4 that if k is part of the input, then the min-max k-TSPT is *NP*-hard in the strong sense, which implies that it has no FPTAS unless $NP = P$.

Theorem 5.4. *If k is part of the input, then the min-max k-TSPT is NP-hard in the strong sense.*

Proof. Given a set J of n jobs where each job j_i has a processing time l_i for $i = 1, 2, \ldots, n$ and a set M of k processors, the multiprocessor scheduling problem is known to minimize the latest completion time to process all the jobs in J on the k processors, such that there is no overlap between any two consecutive jobs on the same processor. According to Garey and Johnson [35], the multiprocessor scheduling problem is *NP*-complete in the strong sense.

Thus, by a reduction from the multiprocessor scheduling problem, we prove as follows that the min-max k-TSPT with k belonging to the input is *NP*-hard in the strong sense. Given any instance of the multiprocessor scheduling problem, consider a star-shaped tree T with $n+1$ vertices denoted by $V = \{1, 2, \ldots, n+1\}$, where vertex $n+1$ is the common end point of all edges, and is also the depot, and the edge weight $w(i, n+1)$ equals l_i for each vertex $i = 1, 2, \ldots, n$. A fleet of k traveling salesmen are initially located at the common depot $n+1$ to service all the customers in $\{1, 2, \ldots, n\}$, with the objective to minimize the maximum total edge weight of tours of all the k traveling salesmen.

Thus, on one hand, any feasible solution to the above instance of the min-max k-TSPT can be transformed to a feasible schedule for the given instance of the multiprocessor scheduling problem, such that the latest completion time of the latter

schedule is at most half of the maximum total edge weight of tours in the former solution. On the other hand, any feasible schedule for the given instance of the multiprocessor scheduling problem can be transformed to a feasible solution to the above instance of the min-max k-TSPT, such that the maximum total edge weight of tours in the latter solution is at most twice the latest completion time of the former schedule. Hence, we obtain that the given instance of the multiprocessor scheduling problem has a feasible schedule with a latest completion time less than or equal to P, if and only if the optimal solution to the above instance of the min-max k-TSPT has an objective value less than or equal to $2P$. This implies that the min-max k-TSPT is *NP*-hard in the strong sense, if k is part of the input. □

5.4.2 Min-max k-TSPT with multiple depots

The min-max k-TSPT with multiple depots (k-TSPT-MD) is an extension of the min-max k-TSPT, where a depot candidate set $D \subseteq V$ is given, so that a feasible solution to the min-max k-TSPT-MD is defined as a k-subtree tuple $(Q_1, Q_2, ..., Q_k)$ of T with each Q_j, for $1 \leq j \leq k$, containing at least one vertex in D as the depot, and with all subtrees in $(Q_1, Q_2, ..., Q_k)$ covering V. Therefore, we can use $[T, w, k, D]$ to denote an instance of the min-max k-TSPT-MD. Similar to the min-max k-TSPT, by following Definition 5.1, we can assume without loss of generality that any given instance $[T, w, k, D]$ of the min-max k-TSPT-MD is standard, and that vertices in $V = \{1, 2, ..., n\}$ are labeled in a non-increasing order on their depths in T, so that n is the root of T.

To obtain a pseudo-polynomial time exact algorithm for the min-max k-TSPT-MD, we first extend the state of any subtree Q of T by including an additional bit $\gamma(Q)$, where $\gamma(Q) = 1$ if Q includes a vertex in D, and $\gamma(Q) = 0$ otherwise. Consider a vertex $v \in V$, where l and r denote the left and right children of v, respectively. Consider any subtree L of $T^{(l)}$ and any subtree R of $T^{(r)}$. We can still

apply Procedure 5.1 to join L and R to obtain a set $\mathcal{Q}$ of subtrees of $T^{(v)}$. Moreover, we can extend Lemma 5.2 as follows, to determine the value of $\gamma(T_t)$ for each subtree T_t, defined in Step 2 of Procedure 5.1, for $1 \leq t \leq 5$, if T_t is returned in $\mathcal{Q}$ by Procedure 5.1.

(i) If T_1 is added to $\mathcal{Q}$, then $\gamma(T_1) = \gamma(L) \vee \gamma(R)$;

(ii) If T_2 is added to $\mathcal{Q}$, then if $v \in D$, then $\gamma(T_2) = 1$, and otherwise, $\gamma(T_2) = 0$;

(iii) If T_3 is added to $\mathcal{Q}$, then if $v \in D$ or $\gamma(L) = 1$, then $\gamma(T_3) = 1$, and otherwise, $\gamma(T_3) = 0$;

(iv) If T_4 is added to $\mathcal{Q}$, then if $v \in D$ or $\gamma(R) = 1$, then $\gamma(T_4) = 1$, and otherwise, $\gamma(T_4) = 0$;

(v) If T_5 is added to $\mathcal{Q}$, then if $v \in D$, or $\gamma(L) = 1$, or $\gamma(R) = 1$, then $\gamma(T_5) = 1$, and otherwise, $\gamma(T_5) = 0$.

We use Example 5.4 to illustrate how to determine the states of subtrees in $\mathcal{Q}$ returned by Procedure 5.1 for this multiple-depot case.

Example 5.4. Let us follow Example 5.2, and apply Procedure 5.1 to join L and R, where $L = \Phi$ is a subtree of $T^{(3)}$ and $R = T^{(6)}$ is a a subtree of $T^{(6)}$. Accordingly, we obtain $\mathcal{Q} = \{T_1, T_4\}$, where for each subtree in $\mathcal{Q}$, the subgraph induced by $V(T^{(3)})$ equals L, and the subgraph induced by $V(T^{(6)})$ equals R. Suppose $D = \{3, 4\}$ is the depot set. Then, the state of each subtree in $\mathcal{Q}$ can be determined as follows, according to the extension of Lemma 5.2 explained before. First, since L contains no depot in D, then $\gamma(L) = 0$, which implies that the state of L is $(0, 0, 1, 0)$. Similarly, since R contains no depot in D, then $\gamma(R) = 0$, which implies that the state of R is $(0, 1, 0, 0)$. Moreover, since $T_1 \in \mathcal{Q}$, then according to the condition (i) above, we obtain that $\gamma(T_1) = \gamma(L) \vee \gamma(R) = 0$. Since $w(T_1) = 0$, $\alpha^{(4)}(T_1) = 0$, $\beta(T_1) = 0$, and $\gamma(T_1) = 0$, we obtain that the state of T_1 is $(0, 0, 0, 0)$. Since $T_4 \in \mathcal{Q}$ and vertex

$4 \in D$, according to the condition (iv) above, we obtain that $\gamma(T_4) = 1$, which implies that the state of T_4 is $(0, 1, 0, 1)$. □

Accordingly, we can revise Algorithm 5.4 as follows to obtain a dynamic programming for the min-max k-TSPT-MD. In Step 1.b, we change the condition (ii) to if $v = n$, then $\gamma(Q_j) = 1$ for $1 \leq j \leq k$, so as to guarantee that each Q_j contains at least one vertex in D as the depot. Let $\tilde{w} = (w_1, w_2, \ldots, w_{k-1})$, $\tilde{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_k)$, $\tilde{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$, and $\tilde{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_k)$. Then, Step 1.c can be revised as follows: For each $\tilde{w} \in \{0, 1..., U\}^{k-1}$, each $\tilde{\alpha} \in \{0, 1\}^k$, each $\tilde{\beta} \in \{0, 1\}^k$, and each $\tilde{\gamma} \in \{0, 1\}^k$ set $w_k^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = +\infty$ and $\mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = \emptyset$. For each $(Q_1, Q_2, \ldots, Q_k)$ in $\mathcal{H}_2^{(v)}$, do the following steps: (i) Let $(w(Q_1), w(Q_2), \ldots, w(Q_{k-1})) = \tilde{w}$, $(\alpha^{(v)}(Q_1), \alpha^{(v)}(Q_2), \ldots, \alpha^{(v)}(Q_k)) = \tilde{\alpha}$, $(\beta(Q_1), \beta(Q_2), \ldots, \beta(Q_k)) = \tilde{\beta}$, $(\gamma(Q_1), \gamma(Q_2), \ldots, \gamma(Q_k)) = \tilde{\gamma}$. (ii) If $w(Q_k) < w_k^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$, then set $w_2^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = w(Q_k)$, and set $\mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = \{(Q_1, Q_2, \ldots, Q_k)\}$. Accordingly, Step 1.d can be revised as follows: Set $\mathcal{H}^{(v)}$ to be empty. For each $\tilde{w} \in \{0, 1..., U\}^{k-1}$, each $\tilde{\alpha} \in \{0, 1\}^k$, each $\tilde{\beta} \in \{0, 1\}^k$, and each $\tilde{\gamma} \in \{0, 1\}^k$, let $\mathcal{H}^{(v)} = \mathcal{H}^{(v)} \bigcup \mathcal{Q}^{\min}(\tilde{w}, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$.

Similar to the proofs of Lemma 5.4, Lemma 5.5, and Theorem 5.2, we can show that the revised dynamic programming returns an optimal solution to any given instance of the min-max k-TSPT-MD in $O(nU^{2(k-1)})$ time, where U can be chosen as W, because $(T, T, ..., T)$ is a feasible solution. Thus, the running time is pseudo-polynomial if k is a given constant independent of the input size.

Next, we are going to develop an FPTAS for the min-max k-TSPT-MD. Since a feasible solution to the min-max k-TSPT-MD may not necessarily cover each edge of the underlying tree T, then $w(T)/k \leq \text{OPT}$ may not be valid here, where OPT is the optimal objective value to any given instance of the min-max k-TSPT-MD. Thus, to find a valid lower bound on OPT, we use $T(\lambda)$ to denote the subgraph of T obtained by removing all the edges with a weight greater than λ. It can be seen that the instance $[T, w, k, D]$ has a feasible solution that consists only of edges with weights

less than or equal to λ, if, and only if, such λ satisfies that $T(\lambda)$ has less than or equal to k connected components, and that each connected component of $T(\lambda)$ has at least one depot in D. Let us define λ^* as the minimum value of such λ that $T(\lambda)$ has less than or equal to k connected components, and that each connected component of $T(\lambda)$ has at least one depot in D. Since λ^* belongs to $\{w(e) : e \in E\}$, we can first sort edge weights in $\{w(e) : e \in E\}$ in a non-decreasing order, and then apply a binary search to the sorted edge weights in $\{w(e) : e \in E\}$ to find λ^*. Thus, since $|E| \leq n$, it can be seen that the total running time to find λ^* is $O(n \log n)$. Moreover, by definition of λ^*, any feasible solution to $[T, w, k, D]$ must contain an edge whose edge weight is at least λ^*. Thus, we can establish Lemma 5.7 as follows for λ^*.

Lemma 5.7. *For any instance $[T, w, k, D]$ of the min-max k-TSPT-MD, it satisfies that $\lambda^* \leq \text{OPT} \leq n\lambda^*$, where* OPT *is the optimal objective value to $[T, w, k, D]$.*

Proof. According to the definition of λ^*, any feasible solution to $[T, w, k, D]$ must contain an edge e satisfying $w(e) \geq \lambda^*$, which implies that $\lambda^* \leq \text{OPT}$. Moreover, there exists a solution, denoted by $\{Q_1, Q_2, \ldots, Q_k\}$, such that each edge of Q_i for $1 \leq i \leq k$ has a weight less than or equal to λ^*, which implies that $\max_{1 \leq i \leq k}\{w(Q_i)\} \leq |E(T(\lambda^*))|\lambda^* \leq n\lambda^*$. Consider an optimal solution $(\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_k)$ to $[T, w, k, D]$. Since $(Q_1, Q_2, \ldots, Q_k)$ is also a feasible solution to $[T, w, k, D]$, we obtain that

$$\max_{1 \leq i \leq k}\{w(\tilde{T}_i)\} \leq \max_{1 \leq i \leq k}\{w(Q_i)\} \leq n\lambda^*. \tag{5.1}$$

This completes the proof. □

Similar to Algorithm 5.3, we develop Algorithm 5.5 for the min-max k-TSPT-MD as follows. Let $\delta = \epsilon\lambda^*/|E(T)|$. Algorithm 5.5 scales w to w' in Step 1 by setting $w'(e) = \lfloor w(e)/\delta \rfloor$ for each $e \in E(T)$, so as to obtain a scaled instance $[T, w', k, D]$ of the min-max k-TSPT-MD. Thus, according to Lemma 5.7, we know that $n\lambda^*/\delta$ is a valid upper bound of the objective value of an optimal solution to the scaled instance

$[T, w', k, D]$. Thus, Algorithm 5.5 applies the revised dynamic programming with $U = n\lambda^*/\delta$ to $[T, w', k, D]$ in Step 2 to obtain an optimal solution $(Q_1^*, Q_2^*, \ldots, Q_k^*)$ to $[T, w', k, D]$, and returns it in Step 3.

Algorithm 5.5.

Input: *An instance $[T, w, k, D]$ to the min-max k-TSPT-MD, any $\epsilon > 0$, and λ^*.*

Output: *A k-subtree tuple $\{Q_1^*, Q_2^*, \ldots, Q_k^*\}$ where Q_j^* for $j = 1, 2, \ldots, k$ is a subtree of T.*

1. *Let $\delta = \epsilon\lambda^*/|E(T)|$. For each $e \in E(T)$, let $w'(e) = \lfloor w(e)/\delta \rfloor$. Then, a scaled instance $[T, w', k, D]$ can be constructed.*

2. *Let $U = n\lambda^*/\delta$. Apply the revised dynamic programming for the min-max k-TSPT-MD to obtain an optimal solution, denoted by $(Q_1^*, Q_2^*, \ldots, Q_k^*)$, to the instance $[T, w', k, D]$.*

3. *Return $(Q_1^*, Q_2^*, \ldots, Q_k^*)$.* □

Consider the optimal solution $\{Q_1^*, Q_2^*, \ldots, Q_k^*\}$ to $[T, w', k, D]$ returned by the revised dynamic programming applied in Step 2 of Algorithm 5.5. Since $\{Q_1^*, Q_2^*, \ldots, Q_k^*\}$ is also a feasible solution to $[T, w, k, D]$, and since $\lambda^* \leq \text{OPT}$ where OPT denotes the objective value of an optimal solution to $[T, w, k, D]$, by following arguments similar to the proof of Theorem 5.3, we can obtain that $\max_{1 \leq i \leq k}\{w(Q_i^*)\} \leq (1 + \epsilon)\text{OPT}$. Moreover, since T is a tree, we have $|E(T)| \leq n - 1$. It can be seen that the running time of Algorithm 5.5 is $O(n^{4k-3}/\epsilon^{2(k-1)})$. Thus, Algorithm 5.5 is an FPTAS for the min-max k-TSPT-MD, if k is a given constant independent of the input size.

5.4.3 Min-max k-TSPT with demand points

The min-max k-TSPT with demand points (k-TSPT-DP) is an extension of the min-max k-TSPT, where each of the given demand points that needs to be covered, is located either at a vertex in V or on an edge in E. Thus, instead of covering all

vertices in $V(T)$, a feasible solution to the min-max k-TSPT-DP is defined as a k-subtree tuple $(Q_1, Q_2, ..., Q_k)$, where Q_j for $1 \leq j \leq k$ is a subtree of T that contains the depot, such that $\bigcup_{j=1}^{k} V(Q_j)$ needs to cover a given subset J of vertices in V, and that $\bigcup_{j=1}^{k} E(Q_j)$ needs to cover a given subset F of edges in E. Thus, we use $[T, w, k, J, F]$ to denote an instance of the min-max k-TSPT-DP. Similar to the min-max k-TSPT, by following Definition 5.1, we can assume without loss of generality that any given instance $[T, w, k, J, F]$ of the min-max k-TSPT-DP is standard, and that vertices in $V = \{1, 2, ..., n\}$ are labeled in a non-increasing order on their depths in T, so that n is the root and the depot of T.

To develop an FPTAS for the min-max k-TSPT-DP, we can revise Algorithm 5.4 to obtain a dynamic programming by replacing the condition (i) of Step 1.b with the following three conditions: (i-1) if $v \in J$ then $v \in \bigcup_{j=1}^{k} V(Q_j)$, (i-2) if $(v, l) \in F$ then $(v, l) \in \bigcup_{j=1}^{k} E(Q_j)$, and (i-3) if $(v, r) \in F$ then $(v, r) \in \bigcup_{j=1}^{k} E(Q_j)$, to ensure that the given vertex subset J and edge subset F are covered. It is easy to see that the dynamic programming returns an optimal solution to any given instance of the min-max k-TSPT-DP, and has a running time $O(nU^{2(k-1)})$, where U is an upper bound on the optimal objective value to the min-max k-TSPT-DP. Since $(T, \Phi, \Phi, ..., \Phi)$ is a feasible solution, we can choose $U = w(T)$, which implies that the running time of the dynamic programming is $O(nW^{2(k-1)})$. Moreover, based on this dynamic programming, for any given $\epsilon > 0$, by following an approach similar to the one for the min-max k-TSPT-MD we can obtain a $(1+\epsilon)$-approximation scheme for the min-max k-TSPT-DP with a running time $O(n^{4k-3}/\epsilon^{2(k-1)})$, which is an FPTAS when k is a given constant independent of the input size.

Furthermore, if we replace condition (i) of Step 1.b in the dynamic programming in Section 5.4.2 for the min-max k-TSPT-MD, with the three conditions (i-1), (i-2) and (i-3) described above for the min-max k-TSPT-DP, we can obtain an exact algorithm for a variant of the min-max k-TSPT, where both a set of multiple candidate depots

and a set of demand points are given. Following an approach similar to the one for the min-max k-TSPT-MD, for any given $\epsilon > 0$, we can also obtain a $(1+\epsilon)$-approximation scheme for this variant.

5.4.4 Min-sum k-TSPT

The min-sum k-TSPT is to determine a k-subtree tuple $(Q_1, Q_2, ..., Q_k)$, where Q_j for $1 \leq j \leq k$ is a subtree of T that contains the depot, such that $\bigcup_{j=1}^{k} V(Q_j)$ covers all the vertices in V, so as to minimize the total weight of all the subtrees in $(Q_1, Q_2, ..., Q_k)$. Given any instance, denoted by $[T, w, k]$, of the min-sum k-TSPT, since the trees in each feasible solution are all connected, it is easy to see that $(T, \Phi, ..., \Phi)$ is an optimal solution to $[T, w, k]$, with an objective value equal to $w(T)$.

Although the min-sum k-TSPT is trivial for any given constant k, neither any polynomial-time exact algorithm nor any proof of *NP*-hardness is known for its extension with multiple depots. The min-sum k-TSPT-MD aims to determine a k-subtree tuple $(Q_1, Q_2, ..., Q_k)$, where Q_j for $1 \leq j \leq k$ is a subtree of T that contains at least a depot in a given depot set D, such that $\bigcup_{j=1}^{k} V(Q_j)$ covers all the vertices in V, so as to minimize the sum of the total edge weight of the subtrees in $(Q_1, Q_2, ..., Q_k)$. Similar to the min-max k-TSPT, by following Definition 5.1, we can assume without loss of generality that any given instance $[T, w, k, D]$ of the min-sum k-TSPT-MD is standard, and that vertices in $V = \{1, 2, ..., n\}$ are labeled in a non-increasing order on their depths in T, so that n is the root of T.

Based on the dynamic programming for the min-max k-TSPT-MD in Section 5.4.2, we can develop a dynamic programming for the min-sum k-TSPT-MD by revising Step 1.c as follows: For each $\tilde{\alpha} \in \{0,1\}^k$, each $\tilde{\beta} \in \{0,1\}^k$, and each $\tilde{\gamma} \in \{0,1\}^k$, set $w^{\min}(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = +\infty$ and $\mathcal{Q}^{\min}(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = \emptyset$. For each $(Q_1, Q_2, \ldots, Q_k)$ in $\mathcal{H}_2^{(v)}$, do the following steps: (i) Let $(\alpha^{(v)}(Q_1), \alpha^{(v)}(Q_2), \ldots, \alpha^{(v)}(Q_k)) = \tilde{\alpha}$, $(\beta(Q_1), \beta(Q_2), \ldots, \beta(Q_k)) =$

$\tilde{\beta}$, and $(\gamma(Q_1), \gamma(Q_2), \ldots, \gamma(Q_k)) = \tilde{\gamma}$. (ii) If $\sum_k w(Q_k) < w^{\min}(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$, then set $w^{\min}(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = \sum_k w(Q_k)$, and set $\mathcal{Q}^{\min} = \{(Q_1, Q_2, \ldots, Q_k)\}$. Accordingly, Step 1.d can be revised as follows: Set $\mathcal{H}^{(v)}$ to be an empty set. For each $\tilde{\alpha} \in \{0,1\}^k$, each $\tilde{\beta} \in \{0,1\}^k$, and each $\tilde{\gamma} \in \{0,1\}^k$, let $\mathcal{H}^{(v)} = \mathcal{H}^{(v)} \bigcup \mathcal{Q}^{\min}(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$.

Similar to Lemma 5.5, we can show that for each $v = 1, 2, ...n$, there exists a tuple $(Q_1, Q_2, ..., Q_k) \in \mathcal{H}^{(v)}$, obtained in Step 1 of the above dynamic programming for the min-sum k-TSPT-MD, such that $\alpha^{(v)}(Q_j) = \alpha^{(v)}(\widetilde{T}_j^{(v)})$, $\beta(Q_j) = \beta(\widetilde{T}_j^{(v)})$, and $\gamma(Q_j) = \gamma(\widetilde{T}_j^{(v)})$ for $1 \leq j \leq k$, and that $\sum_{j=1}^{k} w(Q_j) \leq \sum_{j=1}^{k} w(\widetilde{T}_j^{(v)})$. Accordingly, similar to the proof of Theorem 5.2, we can obtain that the above dynamic programming returns an optimal solution to any given instance of the min-sum k-TSPT-MD, with a running time $O(2^k n)$, which is polynomial when k is a constant independent of the input size.

5.5 Summary

In this chapter, we have developed a pseudo-polynomial time algorithm for the min-max 2-TSPT through a dynamic programming approach, which provides a positive answer to a research question that has remained open for a decade. Based on this dynamic program, we have further developed a fully polynomial time approximation scheme for the problem. We have conducted preliminary numerical experiments to test the performance of the FPTAS for the min-max 2-TSPT. The results indicate that, for the min-max 2-TSPT, the worse-case performance and the average performance of the FPTAS are better than the traditional local search scheme.

Moreover, we have generalized the pseudo-polynomial time exact algorithm and the FPTAS for the min-max k-TSPT with $k \geq 2$ and other multiple vehicle routing problems. The running time of the FPTAS for the min-max k-TSPT with $k \geq 2$ is exponential to k, and is polynomial only when k is a constant independent of the input size. To tackle this problem, we need to design more sophisticated heuristic for

the min-max k-TSPT with $k \geq 2$.

CHAPTER 6

Improved Approximation Algorithm for the k-depot TSP

6.1 Introduction

The k-depot TSP ($k \geq 2$) is an extension of the single depot TSP, and it aims to determine a set of k tours for the k vehicles located at distinct depots, to serve all the customers on a metric undirected graph so that the total length of the tours is minimized. Based on the tree algorithm, Rathinam et al. [69] developed a 2-approximation algorithm for the k-depot TSP which is the best available approximation result. In the literature [3, 59, 69], it has been suggested that the Christofides' heuristic can be extended for the k-depot TSP and its variants by computing a constrained spanning forest in which each tree contains a distinct depot. However, the worst-case analysis of the extended heuristic is known to be difficult since this requires bounding the length of a minimum perfect matching for vertices having odd degrees in the constrained spanning forest [3, 59]. It thus remains an open question whether a tight approximation ratio of the extended Christofides' heuristic can be found for the k-depot TSP.

Initiated by the open question above, we prove a tight approximation ratio $(2 - 1/k)$ of the extended Christofides' heuristic in this chapter, which implies that the

extended Christofides' heuristic is better than the existing 2-approximation algorithm available in the literature, especially when k is small. The remainder of this chapter is organized as follows. In Section 6.2, we introduce the notation that are used throughout this chapter. In Section 6.3, we describe the extended Christofides' heuristic for the k-depot TSP, and present an example to show that its approximation ratio is not less than $(2 - 1/k)$. We then prove in Section 6.4 that $(2 - 1/k)$ is indeed a tight approximation ratio of the extended Christofides' heuristic. Some results derived in Section 6.4 have been generalized in Section 6.5, and several applications of the extended Christofides' heuristic have been demonstrated in Section 6.6 to develop approximation algorithms for other k-depot vehicle routing problems. Finally, we summarize this chapter in Section 6.7.

6.2 Notation

Given an integer $k \geq 2$, let $G = (V, E)$ with vertex set V and edge set E be a complete graph. Each edge in E has a non-negative length, where lengths are assumed to be symmetric and satisfy the triangle inequality. Let $D \subseteq V$ denote a set of k depots, where each depot has a distinct salesman. Taking vertices, denoted by I, which are not depots to represent customers, the k-depot TSP seeks to minimize the total length of a collection of k tours that the salesmen from D use to visit each customer in I exactly once, where each tour must begin at a distinct depot and return to it.

We recall the definitions of *walk*, *tree*, *rooted tree*, *forest*, *matching*, and *perfect matching* [52]. Thus, a walk, which is denoted by $(v_1 v_2 ... v_t v_{t+1})$ where $t \geq 0$, is *closed* if its start vertex v_1 and end vertex v_{t+1} are the same. A walk with no repeated vertices is called a *path*. A closed walk with no repeated vertices except its start and end vertices is called a cycle. An undirected multigraph is *Eulerian* if the degree of each vertex is even and a connected Eulerian multigraph must always have an

Eulerian closed walk, i.e., a closed walk containing every edge [52].

Given a rooted tree T with root r, a vertex u in T is an *ancestor* of a vertex v in T if u lies on the unique path from r to v. If, in addition, (u, v) is the last edge on the path from r to v, then u is the *parent* of v, and v is a *child* of u.

Given a graph H, consider any subgraph H'. Let $V(H')$ and $E(H')$ denote the vertex and the edge sets of H'. Given lengths of edges in $E(H)$, the length of H' is defined as the total length of its edges (where multiple edges are counted multiply), which is denoted by $\ell(H')$. Similarly, we can define $V(W)$, $E(W)$ and $\ell(W)$ for any walk W of H, and $V(E')$ and $\ell(E')$ for any edge subset E' of H. Further, if $V(E') \subseteq V(H')$, then we use $H' \setminus E'$ and $H' \cup E'$ to denote subgraphs on $V(H')$ with edge subsets $E(H') \setminus E'$ and $E(H') \cup E'$, respectively.

Consider a complete graph H with edge lengths satisfying the triangle inequality. Thus, given any closed walk W of H, we can remove all repeated vertices in W by shortcuts to obtain a cycle with length not longer than $\ell(W)$, such that each vertex of W appears exactly once in the cycle. Moreover, consider any cycle C of H, represented by a vertex sequence $(v_1 v_2 ... v_t v_{t+1})$ with $v_{t+1} = v_1$ and $t \geq 0$. For any $V' = \{v_{i_1}, v_{i_2}, ..., v_{i_n}\} \subseteq V(C)$, where $1 \leq i_1 < i_2 < ... < i_n \leq t$, if n is even, then edge subsets $M_1 = \{(v_{i_{2j-1}}, v_{i_{2j}}) : 1 \leq j \leq n/2\}$ and $M_2 = \{(v_{i_{2j}}, v_{i_{2j+1}}) : 1 \leq j \leq n/2\}$ embedded in C are two disjoint perfect matchings in the complete graph on V' and by the triangle inequality, the shorter of M_1 and M_2 is not longer than $\ell(C)/2$.

Finally, for a given k-depot TSP instance, which we denote by (G, D), a *cycle partition* is a collection of cycles of G such that each vertex of G appears exactly once in the cycles. Thus, a *feasible* solution for an instance (G, D) is a cycle partition of k cycles such that each cycle contains exactly one depot in D. An *optimal* solution is a feasible solution with the shortest length, and is denoted by $\mathcal{C}^*$ throughout this work.

6.3 The Extended Christofides' Heuristic

In order to adapt the Christofides' heuristic for the k-depot TSP, it is natural to define a constrained spanning forest as follows.

Definition 6.1. Given a k-depot TSP instance (G, D), a constrained spanning forest (CSF) with regard to (w.r.t.) (G, D) is a set of k rooted trees such that each vertex of G appears exactly once in the trees, and each tree contains exactly one depot in D at which it is rooted. A minimum CSF is a CSF with the shortest length.

In the remainder of this paper, we use F^* to denote a minimum CSF w.r.t. (G, D). Note that for every cycle C in an optimal solution $\mathcal{C}^*$, a tree rooted at the unique depot in C is obtained by either deleting an edge from C if $E(C)$ is not empty, or otherwise leaving it unchanged. The collection of such trees forms a CSF w.r.t. (G, D) by Definition 6.1. Thus, taking $e_{\max}(C)$ to denote the longest edge of a cycle C, where $\ell(e_{\max}(C)) = 0$ if $E(C)$ is empty, we have the following upper bound on $\ell(F^*)$:

$$\ell(F^*) \leq \ell(\mathcal{C}^*) - \sum_{C \in \mathcal{C}^*} \ell(e_{\max}(C)). \tag{6.1}$$

As shown in [69], F^* can be computed in $O(|V|^2)$ time since the problem can be transformed to an equivalent minimum spanning tree problem by contracting multiple depots in D into a single vertex. Hence, as we show in Algorithm 6.1, the Christofides' heuristic for the TSP can be extended for the k-depot TSP by computing F^* in step 1.

Algorithm 6.1.

Input: a complete graph $G = (V, E)$ with a depot set $D \subseteq V$ where $|D| = k$

Output: a feasible solution to the k-depot TSP on (G, D)

1: Compute a minimum CSF F^* w.r.t. (G, D).

2: Create a Eulerian multigraph (which may not be connected), by first finding all

the vertices of V that have odd degree in F^*, which constitutes a vertex set denoted by $\text{Odd}(F^*)$, and then compute a minimum perfect matching $M^*(F^*)$ in the complete graph of $\text{Odd}(F^*)$ and add all the edges in $M^*(F^*)$ to F^*.

3: Find a Eulerian closed walk for each connected component of the Eulerian multigraph, and shortcut each repeated vertex in the Eulerian closed walk by triangle inequality to obtain a cycle. For any resulting cycle obtained from the Eularian closed walks, if it has more than one depots, keep any one of the depots, and shortcut all the other depots by triangle inequality. For each depot that is not in any resulting cycle, construct a cycle that consists of the depot only. Denote the resulting k cycles as $\mathcal{C}(F^*)$. Return $\mathcal{C}(F^*)$.

We establish Theorem 6.1 to show the correctness of Algorithm 6.1, and to provide an upper bound on the length of $\mathcal{C}(F^*)$ given by Algorithm 6.1.

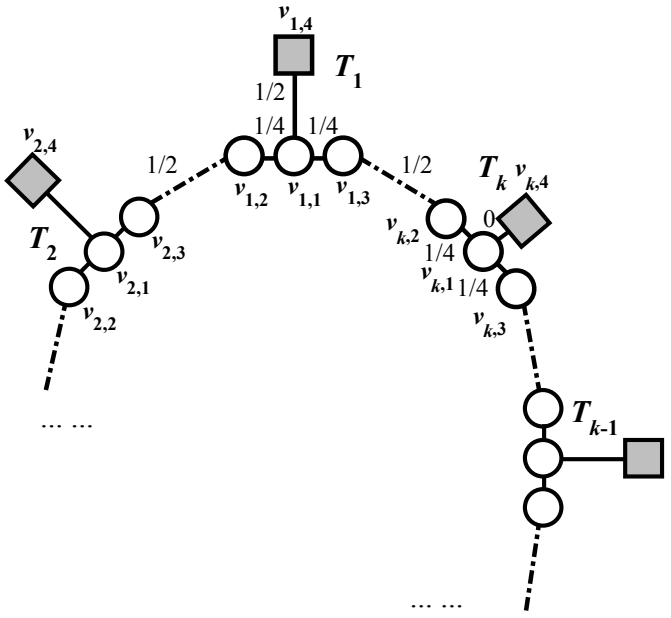
Theorem 6.1. *Given any k-depot TSP instance (G, D), Algorithm 6.1 returns a cycle collection $\mathcal{C}(F^*)$ in $O(|V|^3)$ time, such that $\mathcal{C}(F^*)$ is a feasible solution with $\ell(\mathcal{C}(F^*)) \leq \ell(F^*) + \ell(M^*(F^*))$.*

Proof. Since the number of vertices having odd degrees in F^* is always even, $\text{Odd}(F^*)$ obtained in step 1 of Algorithm 6.1 must have a minimum perfect matching. Thus, adding edges of the minimum perfect matching $M^*(F^*)$ to F^* in step 2 constitutes a Eulerian multigraph, and therefore, each connected component of the multigraph must be Eulerian, and must contain a Eulerian closed walk [52]. Since each of the k trees of F^* is rooted at a distinct depot of D, the Eulerian multigraph obtained in step 2 has at most k connected components with each containing at least one depot. Hence, according to step 3, the collection $\mathcal{C}(F^*)$ returned by Algorithm 6.1 consists of exactly k cycles in which each vertex of G appears exactly once. Moreover, since each tree of F^* contains exactly one depot, each Eulerian closed walk obtained in step 3 contains at least one depot, which implies that each cycle of $\mathcal{C}(F^*)$ must contain

exactly one depot. Therefore, $\mathcal{C}(F^*)$ is a feasible solution by definition. From the construction of the Eulerian multigraph, one can see that, it consists of only those edges in F^* and $M^*(F^*)$. Thus, the total length of the Eulerian multigraph is less than or equal to $\ell(F^*) + \ell(M^*(F^*))$. Since each edge of the Eulerian multigraph (obtained in step 2) appears exactly once in the Eulerian closed walks (obtained in step 3), the total length of these closed walks equals $\ell(F^*)+\ell(M^*(F^*))$, and is greater than or equal to $\ell(\mathcal{C}(F^*))$ due to step 3 and the triangle inequality. Finally, the time complexity of Algorithm 6.1 is $O(|V|^3)$, because computing M^* in step 2 and finding Eulerian closed walks in step 3 can be accomplished in $O(|V|^3)$ time and $O(|V|^2)$ time, respectively [64]. Thus, Theorem 6.1 is proved. □

Although one might expect Algorithm 6.1 to achieve an approximation ratio close to 3/2, we can construct an instance in Example 6.1 to show that the ratio must, in fact, be larger than or equal to $(2 - 1/k)$.

Figure 6.1: An instance of the k-depot TSP with $|D| = k \geq 2$, which shows that the approximation ratio of Algorithm 6.1 with F^* is not smaller than $(2 - 1/k)$.



Example 6.1. For each $k \geq 2$, consider the following complete graph $G = (V, E)$ with $D \subseteq V$ and $|D| = k$. Let V be the union of k disjoint subsets $V_1, V_2, ..., V_k$, where $V_i = \{v_{i,j} : 1 \leq j \leq 4\}$ for $1 \leq i \leq k$. Note that $|V| = 4k$. Let $D = \{v_{i,4} : 1 \leq i \leq k\}$ contain exactly k depots. The customer set is given by $I = V \setminus D$. As shown in Figure 6.1, we first set lengths for the following edges: $\ell(v_{i,1}, v_{i,2}) = \ell(v_{i,1}, v_{i,3}) = 1/4$, for $1 \leq i \leq k$; $\ell(v_{i,1}, v_{i,4}) = \ell(v_{i,2}, v_{i+1,3}) = 1/2$ for $1 \leq i \leq k-1$; $\ell(v_{k,1}, v_{k,4}) = 0$; and $\ell(v_{k,2}, v_{1,3}) = 1/2$. For other edges, lengths are then defined based on the above setting by the length of the shortest path between endpoints. Thus, edge lengths ℓ are symmetric and satisfy the triangle inequality.

We can show that the heuristic solution $\mathcal{C}(F^*)$ returned by Algorithm 6.1 for the instance (G, D) has length equal to $(2 - 1/k)\ell(\mathcal{C}^*)$ and therefore that the ratio is bounded below by $(2 - 1/k)$.

To compute the feasible solution returned by Algorithm 6.1, let us first consider any CSF F w.r.t. (G, D) and derive a lower bound on $\ell(F)$ as follows. Since F consists of k trees, F must have exactly $|V| - k$ edges. Moreover, since each tree in F contains a depot, we have that for each i with $1 \leq i \leq k-1$, F must have at least one edge with one endpoint in $\{v_{i,1}, v_{i,2}, v_{i,3}\}$ (which consists of three customers only) and with its other endpoint in $V \setminus \{v_{i,1}, v_{i,2}, v_{i,3}\}$. According to Figure 6.1, each such edge has length at least $1/2$ and thus, F must have at least $(k-1)$ edges with each having length at least $1/2$. Since each edge other than $(v_{k,4}, v_{k,1})$ has length at least $1/4$, we obtain that $\ell(F) \geq (k-1)/2 + (|V| - k - k)/4 = k - 1/2$.

Next, by definition, the collection $\{T_1, T_2, ..., T_k\}$, as shown by the solid lines in Figure 6.1, where $E(T_i) = \{(v_{i,1}, v_{i,2}), (v_{i,1}, v_{i,3}), (v_{i,1}, v_{i,4})\}$ for $1 \leq i \leq k$, is a CSF w.r.t. (G, D) with length equal to exactly $(k - 1/2)$, and therefore is a minimum CSF w.r.t. (G, D). Thus, we can take $F^* := \{T_1, T_2, ..., T_k\}$ to be the minimum CSF w.r.t. (G, D) obtained in step 1 of Algorithm 6.1, with $\ell(F^*) = k - 1/2$.

We note that $\text{Odd}(F^*)$ here equals V. Consider any perfect matching M for

vertices in V and derive a lower bound on $\ell(M)$ as follows. Let $e_M(v)$ denote the unique edge in M that is incident on v for each $v \in V$. By the fact that $\ell(e_M(v_{k,j})) \geq 1/4$ for $j = 2, 3$ (by Figure 6.1), we can show $\sum_{j=1}^{4} \ell(e_M(v_{k,j})) \geq 1$ by the following arguments. If $\ell(e_M(v_{k,1})) = 0$, then from Figure 6.1, it can be seen that $e_M(v_{k,1})$ must be edge $(v_{k,1}, v_{k,4})$, which implies that neither $e_M(v_{k,2})$ nor $e_M(v_{k,3})$ can be incident on $v_{k,1}$, and therefore, $\ell(e_M(v_{k,j})) \geq 1/2$ for $j = 2, 3$ (according to Figure 6.1). Hence, $\sum_{j=1}^{4} \ell(e_M(v_{k,j})) \geq 1$. Otherwise, $\ell(e_M(v_{k,1})) > 0$, and therefore, $e_M(v_{k,1})$ cannot be incident on $v_{k,4}$. Thus, from Figure 6.1, it can be seen that $\ell(e_M(v_{k,j})) \geq 1/4$ for $j = 1, 4$. This, together with $\ell(e_M(v_{k,j})) \geq 1/4$ for $j = 2, 3$, implies that $\sum_{j=1}^{4} \ell(e_M(v_{k,j})) \geq 1$.

Now, for $1 \leq i \leq k-1$, noting that $\ell(e_M(v_{i,4})) \geq 1/2$ and $\ell(e_M(v_{i,j})) \geq 1/4$ for $1 \leq j \leq 3$ (by Figure 6.1), we can prove that $\sum_{j=1}^{4} \ell(e_M(v_{i,j})) \geq 2$ as follows. If $\ell(e_M(v_{i,4})) > 1/2 + 1/4 = 3/4$, then from Figure 6.1, it can be seen that $\ell(e_M(v_{i,4})) \geq 1/2 + 1/4 + 1/2 = 5/4$ due to the triangle inequality, which implies that $\sum_{j=1}^{4} \ell(e_M(v_{i,j})) \geq 5/4 + 3/4 = 2$. Otherwise, $\ell(e_M(v_{i,4})) \leq 3/4$, and therefore, from Figure 6.1, it can be seen that $e_M(v_{i,4})$ must be in $\{(v_{i,j}, v_{i,4}) : 1 \leq j \leq 3\}$. If $e_M(v_{i,4})$ equals $(v_{i,1}, v_{i,4})$, then $\ell(e_M(v_{i,4})) = \ell(e_M(v_{i,1})) = 1/2$, and neither $e_M(v_{i,2})$ nor $e_M(v_{i,3})$ can be incident on $v_{i,1}$, and therefore, $\ell(e_M(v_{i,j})) \geq 1/2$ for $j = 2, 3$ (by Figure 6.1). Hence, $\sum_{j=1}^{4} \ell(e_M(v_{i,j})) \geq 2$. Otherwise, $e_M(v_{i,4})$ is either $(v_{i,2}, v_{i,4})$ or $(v_{i,3}, v_{i,4})$, which implies that either $e_M(v_{i,2}) = e_M(v_{i,4}) = 1/2 + 1/4 = 3/4$, or $e_M(v_{i,3}) = e_M(v_{i,4}) = 1/2 + 1/4 = 3/4$. This, together with $\ell(e_M(v_{i,j})) \geq 1/4$ for $1 \leq j \leq 3$, implies that $\sum_{j=1}^{4} \ell(e_M(v_{i,j})) \geq 2$.

In summary, since M is a perfect matching for V, we find that

$$\ell(M) = \sum_{i=1}^{k} \sum_{j=1}^{4} \ell(e_M(v_{i,j}))/2 \geq k - 1/2.$$

Moreover, it can be seen that the edge set $\bigcup_{i=1}^{k} \{(v_{i,2}, v_{i,3}), (v_{i,1}, v_{i,4})\}$ is a perfect

matching for V with length exactly equal $(k-1/2)$, and is therefore a minimum perfect matching. Since $\text{Odd}(F^*)$ equals V, we can suppose that $M^*(F^*) := \bigcup_{i=1}^{k}\{(v_{i,2}, v_{i,3}), (v_{i,1}, v_{i,4})\}$ is the minimum perfect matching obtained in step 2 of Algorithm 6.1 for $\text{Odd}(F^*)$, with $\ell(M^*(F^*)) = k - 1/2$.

Accordingly, adding edges of $M^*(F^*)$ to F^*, we can obtain the heuristic solution $\mathcal{C}(F^*) = \{C_1, ..., C_k\}$ returned by step 3 of Algorithm 6.1, where $C_i = (v_{i,4}v_{i,1}v_{i,2}v_{i,3}v_{i,4})$ for $1 \leq i \leq k$. It is easy to see that $\ell(\mathcal{C}(F^*)) = 2k - 1$.

Next, to compute $\ell(\mathcal{C}^*)$ for an optimal solution $\mathcal{C}^*$, we first derive a lower bound on $\ell(\mathcal{C}^*)$ as follows. From Figure 6.1, it can be seen that $\mathcal{C}^*$ must contain at least one edge with length at least $1/2$, which implies that $\sum_{C\in\mathcal{C}^*} \ell(e_{\max}(C)) \geq 1/2$. Due to (6.1) and since $\ell(F^*) = k - 1/2$, we find that $\ell(\mathcal{C}^*) \geq k - 1/2 + 1/2 = k$. Moreover, it is easy to verify that the cycle collection $\{C_1^*, ..., C_k^*\}$, where $C_i^* = (v_{i,4}v_{i,4})$ for $1 \leq i \leq k-1$ and $C_k^* = (v_{k,4}v_{k,2}v_{1,3}v_{1,1}v_{1,2}v_{2,3}v_{2,1}v_{2,2}.... v_{k-1,3}v_{k-1,1}v_{k-1,2}v_{k,3}v_{k,1}v_{k,4})$, is a feasible solution with length equal to exactly k, and therefore is an optimal solution. Thus, $\ell(\mathcal{C}^*) = k$.

Hence, since $\ell(\mathcal{C}(F^*)) = 2k - 1$ and $\ell(\mathcal{C}^*) = k$, we can write $l(\mathcal{C}(F^*)) = (2 - 1/k)\ell(\mathcal{C}^*)$ to show that $(2 - 1/k)$ is a lower bound of the approximation ratio of Algorithm 6.1.

6.4 Approximation Ratio

In this section, we establish Theorem 6.2 to show the tight approximation ratio of Algorithm 6.1.

Theorem 6.2. *Algorithm 6.1 achieves a tight approximation ratio of* $(2 - 1/k)$ *for the* k*-depot TSP.*

As shown in Example 6.1, the approximation ratio of Algorithm 6.1 is at least

$(2-1/k)$. In order to prove Theorem 6.2, we only need to show:

$$\ell(\mathcal{C}(F^*)) \leq (2-1/k)\ell(\mathcal{C}^*). \tag{6.2}$$

Since $\ell(\mathcal{C}(F^*)) \leq \ell(F^*) + \ell(M^*(F^*))$ by Theorem 6.1 and $\ell(F^*)$ can be bounded by (6.1), we only need to derive upper bounds on $\ell(M^*(F^*))$ to verify (6.2). This is shown in (6.3) and (6.4).

To derive the first upper bound on $\ell(M^*(F^*))$, consider each tree T in F^*. By doubling edges of T, we obtain a connected Eulerian multigraph which contains a Eulerian closed walk. After removing repeated vertices of the closed walk by shortcuts, we get a cycle, which we denote by C_T. Note that T contains an even number of vertices in $\mathrm{Odd}(F^*)$, all of which are on the cycle C_T. Thus, C_T must embed a perfect matching M_T for vertices in $V(T) \cap \mathrm{Odd}(F^*)$ with $\ell(M_T) \leq \ell(C_T)/2 \leq \ell(T)$. Combining M_T for all trees T in F^* leads to a perfect matching for vertices in $\mathrm{Odd}(F^*)$, which implies:

$$\ell(M^*(F^*)) \leq \ell(F^*). \tag{6.3}$$

However, the upper bound in (6.3) alone cannot prove (6.2) and we need to establish Lemma 6.1, Lemma 6.2, and Lemma 6.3 below to derive the second upper bound on $\ell(M^*(F^*))$. To do this, we first make use of the notion of an edge set. For any edge subset A of E, let $\mathcal{C}^* \cup A$ denote a graph on V with the edge set that combines $E(\mathcal{C}^*)$ and A. We first establish Lemma 6.1 to show a sufficient condition for $\ell(M^*(F^*)) \leq \ell(\mathcal{C}^*)/2 + \ell(A)$.

Lemma 6.1. *Consider any edge subset A of E. If each connected component of $\mathcal{C}^* \cup A$ contains an even number of vertices in* $\mathrm{Odd}(F^*)$*, then $\ell(M^*(F^*)) \leq \ell(\mathcal{C}^*)/2 + \ell(A)$.*

Proof. Adding edges in A twice to $\mathcal{C}^*$, we can obtain a Eulerian multigraph, denoted

by H, with $\ell(H) \leq \ell(\mathcal{C}^*) + 2\ell(A)$. Note that vertices belonging to the same connected component of H must also belong to the same connected component of $\mathcal{C}^* \cup A$, and vice versa. Thus, if each connected component of $\mathcal{C}^* \cup A$ contains an even number of vertices in $\mathrm{Odd}(F^*)$, then so does each connected component of H. Consider each connected component W of H. Since H is Eulerian, W must also be Eulerian. Thus, since W is connected, it must contain a Eulerian closed walk. By removing repeated vertices of the closed walk by shortcuts, we obtain a cycle denoted by C_W. Note that W contains an even number of vertices in $\mathrm{Odd}(F^*)$, which are all on the cycle C_W. Thus, C_W must embed a perfect matching M_W for vertices in $V(W) \cap \mathrm{Odd}(F^*)$ with $\ell(M_W) \leq \ell(C_W)/2 \leq \ell(W)/2$. Combining M_W for all connected components W of H, we obtain a perfect matching for $\mathrm{Odd}(F^*)$, which implies that $\ell(M^*(F^*)) \leq \ell(H)/2 \leq \ell(\mathcal{C}^*)/2 + \ell(A)$. □

Next, we construct an edge subset A' by the following three steps, such that, as shown later in Lemma 6.2, A' consists of at most $(|\mathcal{C}^*| - 1)$ edges of F^* and satisfies the condition specified for A in Lemma 6.1:

1. Construct a contracted graph, denoted by $[\mathcal{C}^*, E(F^*)]$, in which each cycle of $\mathcal{C}^*$ is represented by a distinct vertex, and vertices representing two different cycles of $\mathcal{C}^*$ are incident with an edge if and only if the two cycles are connected by an edge in $E(F^*)$.

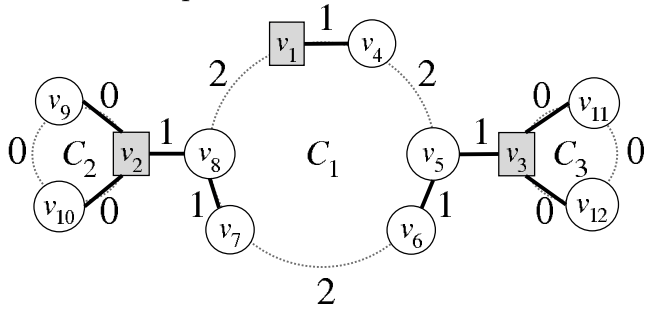
2. Let S denote any forest that consists of a spanning tree in each connected component of the contracted graph $[\mathcal{C}^*, E(F^*)]$.

3. For each edge s in $[\mathcal{C}^*, E(F^*)]$, the two endpoints of s represent two cycles in $\mathcal{C}^*$ which are connected by at least one edge of F^*. Pick exactly one of these edges and denote it by $e(s)$. Define A' as the set of edges $e(s)$ for all $s \in E(S)$.

Example 6.2. Figure 6.2 shows an instance (G, D) of the k-depot TSP with $k = 3$, where vertices v_1, v_2, v_3 (shown by rectangles) are depots, and other vertices (shown by

circles) are customers. The length of each edge shown in Figure 6.2 is indicated by an integer number nearby. For other edges, their lengths are then defined by the length of the shortest path between endpoints. Based on this setting, it is not difficult to see that the three cycles denoted by C_1, C_2, C_3 in Figure 6.2 form an optimal solution $\mathcal{C}^*$ with $\ell(\mathcal{C}^*) = 9$, and that the three trees, shown in solid lines, form an optimal CSF F^* w.r.t. (G, D) with $\ell(F^*) = 5$. Accordingly, to construct A', note that the contracted graph $[\mathcal{C}^*, E(F^*)]$ in step 1 of the construction has three vertices which represent C_1, C_2, C_3, respectively, and has two edges which connect the vertex representing C_1 with the vertices represent C_2 and C_3, respectively. Thus, the forest S defined in step 2 of the construction has only one tree which consists of the two edges of $[\mathcal{C}^*, E(F^*)]$. This implies that $A' = \{(v_2, v_8), (v_3, v_5)\}$ according to step 3 of the construction, because (v_2, v_8) and (v_3, v_5) are the only edges that connect C_1 with C_2 and C_3, respectively. Furthermore, note that $\mathcal{C}^* \cup A'$ has only one connected component, which must contain an even number of vertices in $\text{Odd}(F^*)$. Thus, by Lemma 6.1, we know that $\ell(M^*(F^*)) \leq \ell(\mathcal{C}^*)/2 + \ell(A') \leq 9/2 + 2 = 13/2$. In fact, it can be seen from Figure 6.2 that $M^*(F^*) = \{(v_1, v_4), (v_9, v_{10}), (v_{11}, v_{12}), (v_2, v_7), (v_3, v_6)\}$ with length equal to 5.

Figure 6.2: Example to illustrate the construction of A'.



Using the A' thus constructed, we can establish the second Lemma 6.2.

Lemma 6.2. *(i) A' consists of at most $(|\mathcal{C}^*| - 1)$ edges of F^*. (ii) Each connected component of $\mathcal{C}^* \cup A'$ contains an even number of vertices in* $\text{Odd}(F^*)$.

Proof. To prove (i), note that forest S (in step 2 of the construction of A') has the same number of vertices as the contracted graph $[\mathcal{C}^*, E(F^*)]$, which has at most $|\mathcal{C}^*|$ vertices. Thus, S has at most $(|\mathcal{C}^*|-1)$ edges. According to step 3 of the construction, A' has the same number of edges as S, and A' is a subset of $E(F^*)$. Thus, (i) is proved.

To prove (ii), consider any two vertices u and v belonging to the same connected component of F^*. We know that $\mathcal{C}^*$ has exactly one cycle, which we denote by C_u, that contains u, and exactly one cycle, which we denote by C_v, that contains v, where C_u and C_v are possibly identical. In any case, vertices that represent C_u and C_v in the contracted graph $[\mathcal{C}^*, E(F^*)]$ must belong to the same connected component. This implies u and v belong to the same connected component of $\mathcal{C}^* \cup A'$ in view of the construction of A'. Therefore, since each connected component of F^* contains an even number of vertices in $\mathrm{Odd}(F^*)$, each connected component of $\mathcal{C}^* \cup A'$ must also contain an even number of vertices in $\mathrm{Odd}(F^*)$. □

For the edge subset A' constructed above, we then establish Lemma 6.3 to derive an upper bound on the length of each edge in A'.

Lemma 6.3. *For each edge $(u, v) \in A'$, where u is the parent of v in the unique tree $T_{(u,v)}$ of F^* that contains (u, v), consider the unique cycle C_v in $\mathcal{C}^*$ that contains v. Then, (i) there exists an edge $e \in E(C_v)$ such that $F^* \setminus \{(u, v)\} \cup \{e\}$ is a CSF w.r.t. (G, D), and (ii) $\ell(u, v) \leq \ell(e_{\max}(C_v))$.*

Proof. Since F^* is a minimum CSF w.r.t. (G, D), (ii) can be derived from (i) directly.

To prove (i), let us consider the edge (u, v) first. According to the construction of A', since (u, v) is in A', vertices u and v must belong to two different cycles in $\mathcal{C}^*$. Thus, u is not on C_v.

Consider $T_{(u,v)} \setminus \{(u, v)\}$, which is a forest consisting of two trees, with one tree denoted by T_u containing u, and the other tree denoted by T_v containing v. Note that the only depot contained in $T_{(u,v)}$ is the root of $T_{(u,v)}$, and therefore is contained in T_u

but not in T_v because u is the parent of v in $T_{(u,v)}$. Thus, noticing that C_v contains v and a depot, and that T_v contains v but no depot, we obtain that there must exist an edge e of C_v such that e has exactly one endpoint in $V(T_v)$. Accordingly, let p denote the other endpoint of e, which is not in $V(T_v)$.

Consider a forest F_u defined as $F^* \setminus \{T_{(u,v)}\} \cup \{T_u\}$. Since p is not in $V(T_v)$ and F^* is a CSF w.r.t. (G, D), F_u must contain a unique tree T_p that contains p. Using the edge e to connect T_p and T_v, we can obtain a tree denoted by T'. It can be seen that $F_u \setminus \{T_p\} \cup \{T'\}$ and $F^* \setminus \{(u, v)\} \cup \{e\}$ are the same.

Thus, to prove (i), we only need to show that $F_u \setminus \{T_p\} \cup \{T'\}$ is a CSF w.r.t. (G, D). Note that F^* consists of k trees, which implies that F_u consists of k trees, and so does $F_u \setminus \{T_p\} \cup \{T'\}$. Moreover, since each tree in F^* contains a depot in D, and T_u contains a depot in D, each tree in F_u contains a depot in D. Thus, T_p contains a depot in D, and so does T'. This implies that each tree in $F_u \setminus \{T_p\} \cup \{T'\}$ contains a depot in D, which can be assigned as the root. Finally, it can be seen that each vertex not in $V(T_v) \cup V(T_p)$ must appear exactly once in trees of $F_u \setminus \{T_p\}$. Since $V(T')$ equals $V(T_v) \cup V(T_p)$, each vertex in V must appear exactly once in trees of $F_u \setminus \{T_p\} \cup \{T'\}$. Therefore, according to Definition 6.1, $F_u \setminus \{T_p\} \cup \{T'\}$ is a CSF w.r.t. (G, D), and so is $F^* \setminus \{(u, v)\} \cup \{e\}$. Hence, (i) is proved. □

Let $\ell_{\max} := \max_{C \in \mathcal{C}^*} \ell(e_{\max}(C))$. Noticing that $|\mathcal{C}^*| = k$, from (i) of Lemma 6.2 and (ii) of Lemma 6.3, we obtain that $\ell(A') \leq (k-1)\ell_{\max}$. This, together with Lemma 6.1 and (ii) of Lemma 6.2, implies that

$$\ell(M^*(F^*)) \;\; \leq \;\; \ell(\mathcal{C}^*)/2 + \ell(A') \leq \ell(\mathcal{C}^*)/2 + (k-1)\ell_{\max}, \tag{6.4}$$

which gives the second upper bound on $\ell(M^*(F^*))$.

We can now use (6.3) and (6.4) to prove the main result of this work.

Proof of Theorem 6.2. To show that Algorithm 6.1 has a tight approximation ratio of

$(2-1/k)$, we only need to show (6.2) as follows. On the one hand, from Theorem 6.1, (6.4), (6.1), and $\ell_{\max} \leq \sum_{C\in\mathcal{C}^*} \ell(e_{\max}(C))$, we know:

$$\ell(\mathcal{C}(F^*)) \leq 3\ell(\mathcal{C}^*)/2 + (k-2)\ell_{\max}. \tag{6.5}$$

On the other hand, from Theorem 6.1, (6.3), (6.1), and and $\ell_{\max} \leq \sum_{C\in\mathcal{C}^*} \ell(e_{\max}(C))$, we have:

$$\ell(\mathcal{C}(F^*)) \leq 2\ell(F^*) \leq 2\ell(\mathcal{C}^*) - 2\sum_{C\in\mathcal{C}^*} \ell(e_{\max}(C)) \leq 2\ell(\mathcal{C}^*) - 2\ell_{\max}. \tag{6.6}$$

By (6.5) and (6.6), we obtain:

$$\ell(\mathcal{C}(F^*)) \leq \min\{3\ell(\mathcal{C}^*)/2 + (k-2)\ell_{\max}, 2\ell(\mathcal{C}^*) - 2\ell_{\max}\}. \tag{6.7}$$

Thus, if $3\ell(\mathcal{C}^*)/2+(k-2)\ell_{\max} \leq 2\ell(\mathcal{C}^*)-2\ell_{\max}$, then $(2k)\ell_{\max} \leq \ell(\mathcal{C}^*)$, which, together with (6.7) and $k \geq 2$, implies that $\ell(\mathcal{C}(F^*)) \leq 3\ell(\mathcal{C}^*)/2+(k-2)\ell_{\max} \leq (2-1/k)\ell(\mathcal{C}^*)$. Otherwise, $2\ell(\mathcal{C}^*) - 2\ell_{\max} \leq 3\ell(\mathcal{C}^*)/2 + (k-2)\ell_{\max}$, then $\ell(\mathcal{C}^*) \leq (2k)\ell_{\max}$, which, together with (6.7), implies that $\ell(\mathcal{C}(F^*)) \leq 2\ell(\mathcal{C}^*) - 2\ell_{\max} \leq (2-1/k)\ell(\mathcal{C}^*)$. Hence, (6.2) is proved, and the proof of Theorem 6.2 is completed. □

6.5 Extensions

In Section 6.4, we have proved that the extended Christofides' heuristic for the k-depot TSP, which is shown in Algorithm 6.1, achieves a tight approximation ratio of $(2-1/k)$. Our analysis relies on developing an upper bound on the length $\ell(M^*(F^*))$ in (6.4) derived from Lemma 6.1, Lemma 6.2, and Lemma 6.3 which use a minimum CSF F^*. This naturally suggests CSFs other than F^* might be useful for further improvements. With this in view, we extend the first two lemmas here to provide insights for possible future work.

Lemma 6.4. *Consider any edge subset A of E. For any cycle partition $\mathcal{C}$ and any CSF F w.r.t. (G, D), let $M^*(F)$ denote a minimum perfect matching for vertices in* $\mathrm{Odd}(F)$. *If each connected component of $\mathcal{C} \cup A$ contains an even number of vertices in* $\mathrm{Odd}(F)$, *then $\ell(M^*(F)) \leq \ell(\mathcal{C})/2 + \ell(A)$.*

Proof. For any edge subset A of E, let $\mathcal{C} \cup A$ denote a graph on V with the edge set that combines $E(\mathcal{C})$ and A. If each connected component of $\mathcal{C} \cup A$ contains an even number of vertices in $\mathrm{Odd}(F)$, by replacing $\mathcal{C}^*$ and F^* with $\mathcal{C}$ and F in the proof of Lemma 6.1, it can be shown that $\ell(M^*(F)) \leq \ell(\mathcal{C})/2 + \ell(A)$. □

Lemma 6.2 was derived for a particular edge subset A'. To generalize it, we first extend the notion given by $[\mathcal{C}^*, E(F^*)]$ used in the construction of A'. For any cycle partition $\mathcal{C}$ and any edge subset E', let $[\mathcal{C}, E']$ denote the contracted graph in which each cycle of $\mathcal{C}$ is represented by a distinct vertex, and where vertices representing two different cycles of $\mathcal{C}$ are incident with an edge if and only if the two cycles are connected by an edge in E'. Thus, according to the construction of A' and Lemma 6.2, it is easy to verify the following properties of A': (i) A' is a subset of $E(F^*)$; (ii) each edge in A' connects two different cycles of $\mathcal{C}^*$; (iii) each pair of cycles in $\mathcal{C}^*$ is connected by at most one edge in A'; (iv) each connected component of $\mathcal{C}^* \cup A'$ contains an even number of vertices in $\mathrm{Odd}(F^*)$; and (v) the contracted graph $[\mathcal{C}^*, A']$ is a forest.

Moreover, for any CSF F w.r.t. (G, D) and any cycle partition $\mathcal{C}$, we can construct an edge subset A which has similar properties as A' as follows:

1. Construct a contracted graph, denoted by $[\mathcal{C}, E(F)]$, in which each cycle of $\mathcal{C}$ is represented by a distinct vertex, and vertices representing two different cycles of $\mathcal{C}$ are incident with an edge if and only if the two cycles are connected by an edge in $E(F)$.

2. Let S denote any forest that consists of a spanning tree in each connected component of the contracted graph $[\mathcal{C}, E(F)]$.

3. For each edge s in $[\mathcal{C}, E(F)]$, the two endpoints of s represent two cycles in $\mathcal{C}$ which are connected by at least one edge of F. Pick exactly one of these edges and denote it by $e(s)$. Define A as the set of edges $e(s)$ for all $s \in E(S)$.

In the following lemma, we can show that, the edge subset A satisfies properties similar to A'.

Lemma 6.5. *(i) A is a subset of $E(F)$; (ii) each edge in A connects two different cycles of $\mathcal{C}$; (iii) each pair of cycles in $\mathcal{C}$ is connected by at most one edge in A; (iv)each connected component of $\mathcal{C} \cup A$ contains an even number of vertices in* $\mathrm{Odd}(F)$*; and (v) the contracted graph $[\mathcal{C}, A]$ is a forest.*

Proof. By construction of A, properties (i),(ii) and (iii) are straightforward. By replacing $\mathcal{C}^*$ and F^* with $\mathcal{C}$ and F in the proof of Lemma 6.2, property (iv) can be proved. Finally, in step 2 of construct of A, since S is a forest, then we can conclude that the contracted graph $[\mathcal{C}, A]$ is also a forest. □

6.6 Applications

In the literature, the Christofides' heuristic for the TSP has been extensively applied in the development of approximation algorithms for various single-depot vehicle routing problem, such as the stacker crane problem (SCP) [33], the clustered TSP [41], the capacitated vehicle routing problem (CVRP) [56], and etc. In this section, we are going to illustrate applications of the extended Christofides' heuristic to develop constant ratio approximation algorithms for several multiple depot vehicle routing problems, including the k-depot SCP, the clustered k-depot TSP, and the k-depot CVRP.

6.6.1 The k-depot stacker-crane problem

The Stacker-crane Problem (SCP) is defined on a mixed graph $G = (V, Z \bigcup E)$, where Z is a set of directed arcs, and E is a set of undirected edges. The problem determines the shortest tour which include each arc of Z at least once. Thus, given a depot set $D \subseteq V$, where $|D| = k$, the k-depot stacker-crane problem (k-depot SCP) is to determine the shortest collection of tours that start and end at distinct depots which visit each arc of Z at least once. Frederickson et al. [33] provided two heuristics for this problem, known as LARGEARCS and SMALLARCS. The shorter one of the solutions returned by these two heuristics achieves an approximation ratio of 9/5. Here, we show it is possible to extend these heuristics to solve the k-depot SCP, based on results obtained for the k-depot TSP.

Without loss of generality, assume that arcs of Z are not incident to any depot D (otherwise, we can construct an equivalent instance by making a copy of each depot and move all edges incident to the depot to its copies, such that the resulting instance satisfies the assumption). We use $\mathcal{C}^*$ to denote the optimum solution to the k-depot SCP.

For the LARGEARCS heuristic in Frederickson et al. [33], a minimum bipartite matching between heads and tails of arcs of Z is inserted into G, connected components of the resulting graph are contracted to obtain a contracted graph, and then a minimum spanning tree of the contracted graph is constructed. To solve the k-depot SCP, we construct a minimum CSF F^* of the contracted graph. It is easy to verify that the length of F^* is not longer than $[\ell(\mathcal{C})^* - \ell(Z)]$. Following the same steps of LARGEARCS in Frederickson et al. [33], we are able to construct a Eulerian graph and obtain a solution to the k-depot SCP with length not longer than $[3\ell(\mathcal{C}^*) - 2\ell(Z)]$.

The SMALLARCS heuristic in Frederickson et al. [33], contracts arcs of Z to obtain a contracted graph G', and then applies Christofides' heuristic to obtain a solution to the TSP on G'. To solve the k-depot SCP, we take G' to include depots

of D, and apply the extended Christofides' algorithm on the instance (G', D) to obtain a solution $\mathcal{C}'$ to the k-depot TSP. By Theorem 6.2, we know that $\ell(\mathcal{C}') \leq (2 - 1/k)[\ell(\mathcal{C}^*) - \ell(Z)]$. Following the same steps in SMALLARCS, we can constrct a Eulerian graph and obtain a solution to the k-depot SCP with a length not longer than $[(2 - 1/k)\ell(\mathcal{C}^*) + 1/k\ell(Z)]$.

It can be seen that, the shorter of the solutions returned by LARGEARCS and SMALLARCS achieves a performance ratio of $(2 - 1/(2k + 1))$, which equals to $9/5$ when $k = 2$. Consider the SMALLARCS heuristic for the k-depot SCP. As a comparison, if the tree algorithm developed by Rathinam et al. [69] is applied to obtain a solution $\mathcal{C}'$ to the k-depot TSP rather than the extended Christofides' heuristic, we are able to construct a solution to the k-depot SCP with length not longer than $2\ell(\mathcal{C}^*)$. Therefore, with the same LARGEARCS heuristic for the k-depot SCP, the shorter solution returned by LARGEARCS heuristic and tree algorithm based SMALLARCS heuristic achieves an approximation ratio of 2. Thus, when applied to the k-depot SCP, the extended Christofides' heuristic proposed in this chapter leads to a better approximation algorithm for the k-depot SCP than the tree algorithm.

6.6.2 The clustered k-depot TSP

Given a collection of clusters that forms a partition of the vertex set, the clustered TSP determines the shortest tour that visits every vertex at least once and vertices of each cluster consecutively. Thus, given a depot set D with $|D| = k$, the clustered k-depot TSP determines the shortest collection of k tours which start and end at distinct depots and visit every vertex at least once and vertices of each cluster consecutively in a tour. Guttmann-Beck et al. [41] proposed several approximation algorithms which solve the clustered TSP and achieve constant performance ratios. Their approaches can be extended to solve the clustered k-depot TSP.

For example, consider the case with given start and end vertices for each cluster.

In Guttmann-Beck et al. [41], a 21/11-approximation scheme is used to solve the corresponding variant of clustered TSP. The first step of their method is to determine a Hamiltonian path for each cluster with the start and the end vertices of the cluster as its endpoints. The second step is to solve the stacker-crane problem on the graph with directed arcs from the start to the end vertices of the clusters, to obtain a tour which can be transformed to a solution of the clustered TSP by replacing directed arcs with the corresponding Hamiltonian paths of the first step. To solve the clustered k-depot TSP, we solve the k-depot SCP in step 2 with the approximation algorithm provided in the previous subsection. Let $\mathcal{C}^*$ denote the optimum solution to the clustered k-depot TSP, X denote the set of edges of the paths of $\mathcal{C}^*$ through each cluster, Y denote the set of edges in $\mathcal{C}^*$ but not in X, and Z denote the set of edges of directed arcs from the start vertices to the end vertices in all clusters. Following a similar analysis as in Guttmann-Beck et al. [41], one can find that the length $\ell(\mathcal{P})$ of Hamiltonian paths of the first step is not longer than $\min\{2\ell(X) - \ell(Z), 3/2\ell(X) + 1/2\ell(Z)\}$, and the length of the solution $\mathcal{C}_{scp}$ to the k-depot SCP of the second step is not longer than $\min\{3\ell(Y) + \ell(Z), (2 - 1/k)\ell(Y) + 2\ell(Z)\}$. Thus the length of the solution to the clustered k-depot TSP, which equals $(\ell(\mathcal{P}) - \ell(Z) + \ell(C_{SCP}))$, is not longer than $(2 - 1/(4k + 3))\ell(\mathcal{C}^*)$, which implies that there exists a $(2 - 1/(4k + 3))$-approximation algorithm for the clustered k-depot TSP. Similarly, as a comparison, when a tree algorithm by Rathinam et al. [69] is applied in the second step to obtain a solution to the k-depot SCP, the length of the returned solution is not longer than $\min\{3\ell(Y) + \ell(Z), 2\ell(Y) + 2\ell(Z)\}$. In this case, the solution returned to the clustered k-depot TSP achieves an approximation ratio of 2. Thus, when applied to the clustered k-depot TSP, the extended Christofides' heuristic proposed in this chapter leads to a better approximation algorithm for the k-depot TSP than the tree algorithm.

6.6.3 The k-depot capacitated vehicle routing problem

The Capacitated Vehicle Routing Problem (CVRP) aims to determine a set of routes starting from and ending up with the same depot for vehicles to service customers, such that for each vehicle, after service Q customers, it must return to the depot for replenishment to restart its trip, so as to minimize the total edge weight of the routes. Li and Simchi-Levi [56] provided an approximation algorithm which achieves an approximation ratio of $[1 + (1 - 1/Q)\alpha]$. Their algorithm mostly relies on an Iterated Tour Partitioning (ITP) procedure, which partitions a traveling salesman tour into segments, such that the first segment contains exactly p customers, and that the customer number in each segment does not exceed Q. Given an approximate TSP solution, the ITP procedure is executed Q times to generate Q segment sets denoted by $\mathcal{S}_p$, for $p = 1, 2, \ldots, Q$, such that the first segment in each $\mathcal{S}_p$ contains exactly p customers, and that the customer number in each segment of $\mathcal{S}_p$ does not exceed Q. Then, the segment set with minimum total edge weight is chosen. By joining both of the end points in each segment of the chosen segment set to the depot, a feasible solution to the CVRP can be obtained. When an α-approximation algorithm is applied to generate an approximate TSP tour, the approximation ratio of the algorithm proposed by Li and Simchi-Levi [56] can be proved to be $[1 + (1 - 1/Q)\alpha]$.

The k-depot Capacitated Vehicle Routing problem (k-depot CVRP) is an extension of the CVRP, which takes into consideration a depot D with $|D| = k$ at each of which a distinct vehicle is located. For each vehicle at a distinct depot, it is required to start from and return to its depot, while, in its trip, it can return to any depot in D for replenishment. Here, we use J to denote the customer set, and without loss of generality, we assume that the customer set J and the depot set D are disjoint. As far as we know, no constant ratio approximation algorithm is available in the literature for the k-depot CVRP.

In the followings, we present a constant ratio approximation algorithm for the

k-depot CVRP, based on the extended Christofides' heuristic for the k-depot TSP, and the splitting procedure, Split(C', p) (Algorithm 3.1) developed in Chapter 3:

Step 1: Apply the extended Christofides' heuristic to find an approximate solution $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ for the k-depot TSP.

Step 2: For each tour C_i in the approximate solution for the k-depot TSP, for $i = 1, 2, \ldots, k$, apply Split(C_i, p) to obtain a new cycle C_i^p, for $p = 1, 2, \ldots, Q$, such that C_i^p covers all customers of C_i with both of its endpoints at the depot of C_i, with the first trip containing p customers, and with no trip containing more than Q customers.

Step 3: For $i = 1, 2, \ldots, k$, let C_i' indicate the cycle that minimizes $\ell(C_i^p)$ among all the cycles C_i^p for $p = 1, 2, \ldots, Q$.

Step 4: Return $\mathcal{C}' = \{C_1', C_2', \ldots, C_k'\}$.

It can be verified that, each cycle C_i' obtained in Step 3, where $1 \leq i \leq k$, starts from and ends up with a distinct depot, and that all the customers are covered by cycles, $C_1', C_2', \ldots, C_k'$, in $\mathcal{C}'$ without violating the capacity of the vehicles. Thus, $\{C_1', C_2', \ldots, C_k'\}$ is a feasible solution to the k-depot CVRP.

Let $\mathcal{C}^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$ denote an optimal solution to the k-depot CVRP. Since $\mathcal{C}^*$ is also a feasible solution to the k-depot TSP, and since the extended Christofides' heuristic used in Step 1 has an approximation ratio of $(2 - 1/k)$, we obtain that $\ell(\mathcal{C}) \leq (2 - 1/k)\ell(\mathcal{C}^*)$. Moreover, according to Lemma 3.1, for $i = 1, 2, \ldots, k$, we have

$$\ell(C_i') = \min_{1 \leq p \leq Q} \ell(C_i^p) \leq \ell(C_i) + \frac{2}{Q} \sum_{v \in J(C_i)} \ell_{\min}(v, D),$$

where $J(C_i)$ denotes the set of customers on C_i, and $\ell_{\min}(v, D)$ is defined as the

distance between v and its closest depot in D. Thus,

$$\sum_{i=1}^{k} \ell(C_i') \leq \sum_{i=1}^{k} \ell(C_i) + \frac{2}{Q} \sum_{v \in J} \ell_{\min}(v, D). \tag{6.8}$$

Consider any cycle C_i^* in the optimum solution $\mathcal{C}^*$, where $1 \leq i \leq k$. Let $\eta(i)$ denote the total number of times that C_i^* leaves a depot in D. Thus, C_i^* contains exactly $\eta(i)$ subsegments between two consecutive depots, denoted by $S_{i,1}^*, S_{i,2}^*, \ldots, S_{i,\eta(i)}^*$. For $1 \leq j \leq \eta(i)$, let v be any customer contained in $S_{i,j}^*$. Since $S_{i,j}^*$ contains two depots in D, by the triangle inequality, we have $2\ell_{\min}(v, D) \leq \ell(S_{i,j}^*)$. Since each $S_{i,j}^*$ contains at most Q customers, and since $\sum_{j=1}^{\eta(i)} \ell(S_{i,j}^*) = \ell(C_i^*)$, we have

$$\sum_{v \in J(C_i^*)} \frac{\ell_{\min}(v, D)}{Q} = \sum_{j=1}^{\eta(i)} \sum_{v \in J(S_{i,j}^*)} \frac{\ell_{\min}(v, D)}{Q} \leq \sum_{j=1}^{\eta(i)} \frac{1}{2} \ell(S_{i,j}^*) = \frac{1}{2} \ell(C_i^*),$$

which implies that

$$\frac{2}{Q} \sum_{v \in J} \ell_{\min}(v, D) \leq \sum_{i=1}^{k} \ell(C_i^*) = \ell(\mathcal{C}^*).$$

Together with $\ell(\mathcal{C}) \leq (2 - 1/k)\ell(\mathcal{C}^*)$, it is obtained that $\sum_{i=1}^{k} \ell(C_i') \leq (3 - 1/k)\ell(\mathcal{C}^*)$. Thus, our algorithm proposed for the k-depot CVRP achieves an approximation ratio of $(3 - 1/k)$. As a comparison, if $\mathcal{C}$ in Step 1 is constructed by doubling each edge of the minimum CFS F^* rather than applying the extended Christofides' heuristic, we will obtain an approximation algorithm with an approximation ratio of 3, which is worse than the proposed $(3 - 1/k)$-approximation algorithm above based, especially when k is small.

6.7 Summary

In this chapter, we have proved that an extended Christofides' heuristic achieves a tight approximation ratio of $(2 - 1/k)$ for the k-depot TSP, which is better than

the existing 2-approximation algorithm in the literature. We have also demonstrated that the extended Christofides' algorithm can be applied to develop approximation algorithms for other multiple-depot vehicle routing problems. Moreover, since the best possible approximation ratio for the k-depot TSP is 3/2 unless there exists a better approximation algorithm for the TSP than the Christofides' heuristic, it sill remains an open question on whether or not there exists a tight 3/2 approximation algorithm for the k-depot TSP. Therefore, we have also generalized our results for the extended Christofides' heuristic in this chapter to prepare for future attempt to address this open question.

CHAPTER 7

Conclusions

In this thesis, we have studied the approximability of three categories of VRPs, namely, the min-max PCP, the min-max k-TSPT, and the k-depot TSP, by either designing improved constant ratio approximation algorithms and/or proving the inapproximability bounds.

For the four typical variants of the min-max PCP, where vehicles have either unlimited or limited capacities, and start from either the same depot or different depots, we have presented their first approximation algorithms and their first approximation hardness results. One possible research direction is to improve the worst-case approximation ratios, or to derive tighter inapproximability bounds. Moreover, it may be of more practical value to develop more sophisticated heuristics that can exhibit good average performance in terms of solution qualities and running times. Taking the solutions output by the approximation algorithms developed in this thesis as initial solutions, one can apply various neighborhood search schemes to obtain better solutions by local search, simulated annealing, or other meta-heuristics.

For the min-max k-TSPT, we have in this thesis developed a pseudo-polynomial time algorithm and an FPTAS. However, the running times of the algorithms developed are exponential to the number of vehicles, k, and is polynomial time only if k is a given constant not included in the problem instance. Moreover, when k can be

arbitrarily large, we have shown that the k-TSPT is strongly *NP*-hard. Therefore, it may be of more practical value to develop constant ratio approximation algorithms or heuristics for this general version of the problem. One possible approach for developing a heuristic for the k-TSPT is to use the FPTAS for the min-max 2-TSPT, as follows. Given any feasible solution to the k-TSPT with $k \geq 3$, which is a set of k subtrees, the subtree with the greatest total edge weight is referred to as the "critical" subtree, in the sense that it decides the objective value of the feasible solution. Thus, we can adopt the FPTAS for the min-max 2-TSPT on this "critical" subtree and any other subtree in the current feasible solution to see whether or not the maximum weight of these two subtrees can be reduced. This improvement procedures can be continued until there is no further improvement. Since the running time of the FPTAS for the min-max 2-TSPT is $O(n^3/\epsilon^2)$, each iteration of the proposed heuristic for the k-TSPT is in polynomial time. In our future work, we will conduct extensive numerical experiments to evaluate the performance of this heuristic.

For the k-depot TSP, we have developed a tight $(2-1/k)$-approximation algorithm which is better than the 2-approximation algorithm available in the literature. As mentioned previously, the best possible approximation ratio for the k-depot TSP is 3/2, unless there is an approximation algorithm superior to the Christofides' heuristic for the TSP in a metric undirected graph. Therefore, the question still remains open as to whether or not the k-depot TSP has a 3/2-approximation algorithm. One possible approach to improve the extended Christofides' heuristic is to use a CFS rather than F^* in the first step. The results that we have generalized in Section 6.5 for arbitrary CFSs may be useful in the analysis of the approximation ratio for such approximation algorithms.

Moreover, there are other variants of the k-depot TSP whose approximation algorithms may be improved by certain extended Christofides' heuristics. For example, consider the generalized k-depot TSP, in which $m \geq k$ traveling salesmen are located

at distinct depots, and only k of them are selected to service all the customers. Malik et al. [59] have developed the following 2-approximation algorithm based on the tree algorithm: Step 1: Add a root vertex r to connect to all the vertices corresponding to traveling salesmen with zero edge weight, and remove all edges between vertices corresponding to traveling salesmen. Step 2-1: Calculate an MST of the result graph, which requires the total degrees of the vertices denoting traveling salesmen to be at most $m + k$. Step 2-2: Remove the m edges between the root r and the vertices denoting traveling salesmen to obtain at most k non-trivial trees. Step 3: Double each edge of the k non-trivial trees and obtain a tour for each of the k traveling salesmen by shortcut. Accordingly, it is natural to extend the Christofides' heuristic for this generalized problem, by adding a perfect matching between all vertices which have odd degrees in the k non-trivial trees constructed in step 3. However, the analysis of approximation ratio for such an extended Christofides' heuristic can be challenging, because the tree obtained in steps 2-1 and 2-2, and used in step 3 is not F^*. However, the results that we have generalized in Section 6.5 for arbitrary CFSs have laid a foundation to tackle the challenges.

Bibliography

[1] Y. Agarwal, K. Mathur, and H.M. Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749, 1989.

[2] S. Anily, J. Bramel, and A. Hertz. A 5/3-approximation algorithm for the clustered traveling salesman tour and path problems. *Operations Research Letters*, 24(1-2):29–35, 1999.

[3] E. M. Arkin, R. Hassin, and A. Levin. Approximations for minimum and minmax vehicle routing problems. *Journal of Algorithms*, 59(1):1–18, 2006.

[4] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Mathematical Programming*, 97(1):43–69, 2003.

[5] T. Asano, N. Katoh, and K. Kawashima. A new approximation algorithm for the capacitated vehicle routing problem on a tree. *Journal of combinatorial optimization*, 5(2):213–231, 2001.

[6] P. Augerat, J.M. Belenguer, E. Benavent, A. Corbéran, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2-3):546–557, 1998.

[7] I. Averbakh and O. Berman. Sales-delivery man problems on treelike networks. *Networks*, 25(2):45–58, 1995.

[8] I. Averbakh and O. Berman. A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree. *Discrete Applied Mathematics*, 68 (1-2):17–32, 1996.

[9] I. Averbakh and O. Berman. $(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective. *Discrete Applied Mathematics*, 75(3):201–216, 1997.

[10] I. Averbakh and O. Berman. Minmax p-traveling salesmen location problems on a tree. *Annals of Operations Research*, 110(1):55–68, 2002.

[11] C. Basnet, L.R. Foulds, and J.M. Wilson. Heuristics for vehicle routing on treelike networks. *Journal of the Operational Research Society*, 50(6):627–635, 1999.

[12] C. Bazgan, R. Hassin, and J. Monnot. Approximation algorithms for some vehicle routing problems. *Discrete Applied Mathematics*, 146(1):27–42, 2005.

[13] J.E. Beasley. Route first–cluster second methods for vehicle routing. *Omega*, 11 (4):403–408, 1983.

[14] J. Bruno and P. Downey. Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM Journal on Computing*, 7:393–404, 1978.

[15] A.M. Campbell, D. Vandenbussche, and W. Hermann. Routing for relief efforts. *Transportation Science*, 42(2):127–145, 2008.

[16] P.R Chandler and M. Pachter. Research issues in autonomous control of tactical UAVs. In *American Control Conference, 1998. Proceedings of the 1998*, volume 1, pages 394–398. IEEE, 1998.

[17] P.R. Chandler, M. Pachter, and D. Swaroop. Complexity in UAV cooperative control. In *American Control Conference, 2002. Proceedings of the 2002*, volume 3, pages 1831–1836. IEEE, 2002.

[18] E. Choi and D.W. Tcha. A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers and Operations Research*, 34(7):2080–2095, 2007.

[19] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University*, 1976.

[20] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1):255–282, 1981.

[21] N. Christofides, E. Benavent, V. Campos, A. Corberan, and E. Mota. An optimal method for the mixed postman problem. In P. Thoft-Christensen, editor, *System Modelling and Optimization, Lecture Notes in Control and Information Sciences*, volume 59, pages 641–649. Springer, Berlin, 1984.

[22] J.F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.S. Sormany. New heuristics for the vehicle routing problem. *Logistics systems: design and optimization*, pages 279–297, 2005.

[23] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[24] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2): 342–354, 1992.

[25] J. Edmonds and E. L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[26] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part i: The Chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

[27] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part ii: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.

[28] G. Even, N. Garg, J. Konemann, R. Ravi, and A. Sinha. Minmax tree covers of graphs. *Operations Research Letters*, 32(4):309–315, 2004.

[29] M. Fischetti, P. Toth, and D. Vigo. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42 (5):846–859, 1994.

[30] M.L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42(4):626–642, 1994.

[31] M.L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[32] L.R Foulds and J.M. Wilson. A variation of the generalized assignment problem arising in the New Zealand dairy industry. *Annals of Operations Research*, 69: 105–114, 1997.

[33] G. N. Frederickson. Approximation algorithms for some postman problems. *Journal of the ACM*, 26(3):538–554, 1979.

[34] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[35] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, NY, USA, 1979.

[36] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[37] M. Gendreau, G. Laporte, and J.Y. Potvin. Metaheuristics for the capacitated VRP. *The vehicle routing problem*, 9:129–154, 2002.

[38] B.E. Gillett and L.R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974.

[39] I. D. Giosa, I. L. Tansini, and I. O. Viera. New assignment algorithms for the multi-depot vehicle routing problem. *Journal of the Operational Research Society*, 53(9):977–984, 2002.

[40] N. Guttmann-Beck, R. Hassin, S. Khuller, and B. Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000.

[41] N. Guttmann-Beck, R. Hassin, S. Khuller, and B. Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000.

[42] S. Hamaguchi and N. Katoh. A capacitated vehicle routing problem on a tree. *Algorithms and Computation*, LNCS1553:399–407, 1998.

[43] T. Harks, F.G. Konig, and J. Matuschke. Approximation algorithms for capacitated location routing. In *International Conference on Operations Research*, volume 2010, 2010.

[44] J.A Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991.

[45] Y. Hu. *Approximation algorithms for the capacitated vehicle routing problem*. PhD thesis, Simon Fraser University, 2009.

[46] D.S Johnson and K.A Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8(1): 1–14, 1983.

[47] Y. Karuno and H. Nagamochi. 2-approximation algorithms for the multi-vehicle scheduling problem on a path with release and handling times. *Discrete applied mathematics*, 129(2-3):433–447, 2003.

[48] Y. Karuno and H. Nagamochi. An approximability result of the multi-vehicle scheduling problem on a path with release and handling times. *Theoretical computer science*, 312(2-3):267–280, 2004.

[49] Y. Karuno, H. Nagamochi, and T. Ibaraki. Vehicle scheduling on a tree to minimize maximum lateness. *Journal of the Operations Research Society of Japan-Keiei Kagaku*, 39(3):345–355, 1996.

[50] Y. Karuno, H. Nagamochi, and T. Ibaraki. Vehicle scheduling on a tree with release and handling times. *Annals of Operations Research*, 69:193–207, 1997.

[51] Y. Karuno, H. Nagamochi, and T. Ibaraki. Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks. *Networks*, 39(4): 203–209, 2002.

[52] B.H. Korte and J. Vygen. *Combinatorial optimization: theory and algorithms*, volume 21. Springer Verlag, 2008.

[53] A. Kovács and T. Kis. Partitioning of trees for minimizing height and cardinality. *Information processing letters*, 89(4):181–185, 2004.

[54] M. Labbé, G. Laporte, and H. Mercure. Capacitated vehicle routing on trees. *Operations research*, 39(4):616–622, 1991.

[55] G. Laporte and F. Semet. Classical heuristics for the capacitated VRP. *The vehicle routing problem*, pages 109–128, 2002.

[56] C.-L. Li and D. Simchi-Levi. Worst-case analysis of heuristics for multidepot capacitated vehicle routing problems. *INFORMS Journal on Computing*, 2(1): 64–73, 1990.

[57] A. Lucena and J.E. Beasley. Branch and cut algorithms. In *Advances in linear and integer programming*, pages 187–221. Oxford University Press, Inc., 1996.

[58] J.A. Lukes. Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development*, 18(3):217–224, 1974.

[59] W. Malik, S. Rathinam, and S. Darbha. An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Operations Research Letters*, 35(6):747–753, 2007.

[60] R. G. Michael and S. J. David. *Computers and intractability: a guide to the theory of NP-completeness*. WH Freeman & Co., 1979.

[61] H. Nagamochi. Approximating the minmax rooted-subtree cover problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E88-A(5):1335–1339, 2005.

[62] H. Nagamochi and K. Okada. A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree. *Discrete Applied Mathematics*, 140(1-3): 103–114, 2004.

[63] H. Nagamochi and K. Okada. Approximating the minmax rooted-tree cover in a tree. *Information Processing Letters*, 104(5):173–178, 2007.

[64] C. H. Papadimitriou and K. Steiglitz. Approximation algorithms for the traveling salesman problem. In *Combinatorial Optimization: Algorithms and Complexity*, pages 410–419. Courier Dover Publications, 1998.

[65] C.H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

[66] H.N. Psaraftis, M.M. Solomon, T.L. Magnanti, and T.U. Kim. Routing and scheduling on a shoreline with release times. *Management Science*, 36(2):212–223, 1990.

[67] S. Rathinam and R. Sengupta. 3/2-approximation algorithm for a generalized, multiple depot, Hamiltonian path problem. Technical report, University of California, Berkeley, 2007.

[68] S. Rathinam and R. Sengupta. 5/3-approximation algorithm for a multiple depot, terminal Hamiltonian path problem. Technical report, University of California, Berkeley, 2007.

[69] S. Rathinam, R. Sengupta, and S. Darbha. A resource allocation algorithm for multi-vehicle systems with non holonomic constraints. *IEEE Transactions on Automation Sciences and Engineering*, 4(1):98–104, 2006.

[70] S. Rathinama and R. Sengupta. 3/2-approximation algorithm for two variants of a 2-depot Hamiltonian path problem. *Operations Research Letters*, 38(1):63–68, 2010.

[71] J. Renaud, G. Laporte, and F. F. Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers and Operations Research*, 23(3):229–235, 1996.

[72] C.C. Ribeiro and F. Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42(1):41–52, 1994.

[73] D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis, et al. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6:563–581, 1977.

[74] A. Tamir. Fully polynomial approximation schemes for locating a tree-shaped facility: a generalization of the knapsack problem. *Discrete Applied Mathematics*, 87(1-3):229–243, 1998.

[75] P. Toth and D. Vigo. *The vehicle routing problem*, volume 9. Society for Industrial Mathematics, 2002.

[76] J.N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992.

[77] M. Xiao, T. Fukunaga, and H. Nagamochi. FPTASs for some cut problems in weighted trees. *Frontiers in Algorithmics*, pages 210–221, 2010.

[78] L. Xu, Z. Xu, and D.S. Xu. Exact and approximation algorithms for the min-max k-traveling salesmen problem on a tree. *Working paper*, 2011.

[79] Z. Xu and Q. Wen. Approximation hardness of min-max tree covers. *Operations Research Letters*, 38(3):408–416, 2010.

[80] Z. Xu, L. Xu, and C.-L. Li. Approximation results for min-max path cover problems in vehicle routing. *Naval Research Logistics*, 57(8):728–748, 2010.

[81] Z. Xu, L. Xu, and B. Rodrigues. An analysis of the extended Christofides heuristic for the k-depot tsp. *Operations Research Letters*, 39(3):218–223, 2011.

[82] S. Yadlapalli, W. Malik, S. Darbha, and M. Pachter. A lagrangian-based algorithm for a multiple depot, multiple traveling salesmen problem. *Nonlinear Analysis: Real World Applications*, 10(4):1990–1999, 2009.