NEW ALGORITHMS FOR TWO LOGISTICS OPTIMIZATION PROBLEMS

XIAOFAN LAI

Ph.D

The Hong Kong Polytechnic University

2015

The Hong Kong Polytechnic University

Department of Logistics and Maritime Studies

New Algorithms for Two Logistics Optimization Problems

Xiaofan LAI

A thesis submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
May 2015

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)

__LAI Xiaofan____ (Name of student)

ABSTRACT

New Algorithms for Two Logistics Optimization Problems

by

Xiaofan LAI

Logistics optimization plays a critical role for modern companies in minimizing their costs to gain competitive advantage. In this thesis we study two logistics optimization problems, with one to determine an optimal route for a vehicle, which is a problem at operational level, and with the other to jointly decide on facility locations and network connections, which is a problem at both strategic and tactic level. Both of the two problems aim to minimize the total operational cost and have wide applications in transportation industry. For the first problem, its solution can help shipping carriers using a barge to reposition empty containers in a tree-shaped water system. For the second problem, its solution can be used to help transportation companies to construct plans of facility locations combined with vehicle routing or cargo shipping. Since both of the two problems are $\mathcal{NP}$-hard, i.e., there unlikely exists any polynomial time algorithms that can solve them to optimality, it is of great interest to develop efficient or improved approximation algorithms that can produce near-optimal solutions in affordable running time for these two problems.

This thesis comprises two essays and presents newly developed algorithms for these two logistics optimization problems. The problem studied in the first essay is

named the Capacitated Traveling Salesman Problem with Pickup and Delivery on a Tree, and it aims to determine the best route for a vehicle with a finite capacity to transport amounts of a product from pickup points to delivery points on a tree-shaped network, such that the total travel distance of the vehicle is minimized. This problem has several applications in transportation industry and is well-known to be strongly $\mathcal{NP}$-hard. We therefore develop a 2-approximation algorithm that is a significant improvement over the best constant approximation ratio of 5 derived from existing literature. Computational results show that the proposed algorithm also achieves good average performance, with a much shorter running time and better quality solution, over randomly generated instances.

The problem studied in the second essay is named the k-median Steiner Forest Problem that jointly optimizes the opening of at most k facility locations and their connections to the client locations, so that each client is connected by a path to an open facility, with the total connection cost minimized. The problem has wide applications in the transportation and telecommunication industries, but is known to be strongly $\mathcal{NP}$-hard. In the literature, only a 2-approximation algorithm is available, it being based on a Lagrangian relaxation of the problem and using a sophisticated primal-dual schema. Therefore, we develop an improved approximation algorithm using a simple transformation from an optimal solution of a minimum spanning tree problem. Compared with the existing algorithm, our new algorithm not only achieves a better approximation ratio that is easier to be proved, but also guarantees to produce solutions of equal or better quality-up to 50% improvement in some cases. Additionally, for two non-trivial special cases, where either every location contains a client, or all the locations are in a tree-shaped network, we have developed, for the first time in the literature, new algorithms that can solve the problem to optimality in polynomial time.

Publications arising from this thesis

This thesis is mainly based on the following publication and working paper [52, 77]:

- An Improved Approximation Algorithm for the Capacitated TSP with Pickup and Delivery on a Tree, co-authored with Zhou Xu, Andrew Lim and Fan Wang, *Networks*, 63(2):179–195, 2014.

- Improved Algorithms for Joint Optimization of Facility Locations and Network Connections, co-authored with Zhou Xu, accepted by *European Journal of Operational Research*, 2015.

Moreover, the author of this thesis also completed two other research works during his Ph.D study, which result in other papers as follows:

- A Multi-objective Optimization for Green Supply Chain Network Design, co-authored with Fan Wang and Ning Shi, *Decision Support Systems*, 51(2):262–269, 2011.

- Purchasing Transportation Services from Ocean Carriers, co-authored with Zhou Xu, in *Handbook of Ocean Container Transport Logistics: Making Global Supply Chains Effective*, Chung-Yee Lee and Qiang Meng (Ed.), 399–428, Springer, 2015.

However, since the above papers are not related to providing new improved approximation algorithms, we do not include them into this thesis.

ACKNOWLEDGEMENTS

I would like to thank The Hong Kong Polytechnic University (PolyU) for providing me with both the opportunity and perfect environment to pursue my Ph.D degree. At this point in my life I am highly indebted to so many people for their assistance and support during my studies and life here at PolyU.

First and foremost, I would like to extend my sincerest appreciation to my chief supervisor Dr Zhou Xu for his insightful guidance, instructive suggestions and intellectual inspiration during the completion of this thesis. Working with him has become one of the most unforgettable experiences of my life so far – his strict attitude toward research, his acute research sense and his extreme efficiency have made a deep impression on me that will benefit me throughout the rest of my life. Furthermore, I have been honored to have Prof. Liming Liu as my co-supervisor. His encouragement and instruction have benefited me immensely, and without his constant support and help I could not have accomplished all that I have.

My thanks also go to my two co-authors, Prof. Fan Wang and Prof. Andrew Lim, who have contributed to one chapter of my thesis. In particular, Prof. Fan Wang, as supervisor during my Master studies, is always willing to share with me his wealth of knowledge and expertise, and I cannot imagine how my academic career would turn out without his guidance and continuous assistance.

I am grateful to all the other great teachers at PolyU during my Ph.D studies – ones like Dr Pengfei Guo, Dr Li Jiang, Prof. Eric Ngai and Dr Xiaowen Fu, along with so many others whose delightful sharing has equipped me with the necessary

knowledge and skills. Moreover, I am also thankful to those from our department, such as Prof. Hong Yan, Prof. Chung-Lun Li, Prof. Andy Yeung, Dr Meifeng Luo, Dr Hengqing Ye, Dr Yulan Wang, Dr Bibo Yang and Dr Johnny Wan, as well as the staff in the General Office of LMS and FB, such as Ms. Joyce Yip, Ms. Irene Lam, and so on. Their enthusiastic help has played an undoubted role in my Ph.D studies. In particular, special thanks go to Dr Daniel Ng for his generous support and care.

Moreover, I would like to sincerely thank my thesis Board of Examiners chairman Dr Tsz Leung Yip, and the committee members, Prof. Xiangtong Qi and Dr Yanzhi Li, for their time and valuable comments on my thesis.

I am also thankful for all my dear friends during this period of my Ph.D studies, especially the roommates who live together in 1986 and B1310, the classmates who study together in offices NEOC803, NEOC102, W710 and CD405, the fellows who always lunch together from the BRE and CEE Departments, as well as my academic brothers who willingly discuss together and share their knowledge with me. Their names are not listed here due to the page limit. They have all provided me with immense help and support in every respect. I am also grateful to Dr Jinwen Ou and Dr Xian Zheng from Jinan University for sharing with me their experience in academia, as well as their helpful suggestions on my career development and planning.

Finally, I would like to express my heartfelt gratitude and admiration to my family members, especially my grandparents, my parents, my wife and my sister. Their support, unconditional love and tremendous tolerance have smoothed the way for my studies and research.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER 1

Introduction

As market competition becomes progressively fiercer, logistics optimization plays an increasingly critical role for modern companies seeking to gain a competitive edge. The logistics optimization process generally includes three planning level decisions: the strategic level, the tactical level and the operational level [1]. Specifically, the strategic level refers to decisions that have long-term implications for the companies, such as decisions on facility location, ship fleet size and mix. The tactical level relates to the implementation of strategic level decisions and includes decisions that have relatively short-term implications on the companies, such as decisions on service network design, scheduling and network connections. The operational level relates to day-to-day operations of the companies and decisions that are always made with a short-term decision horizon, such as decisions on vehicle routing, cargo selection and cargo shipping. Therefore, logistics optimization problems can be described as a class of problems necessitating decisions involving any combination of the above three planning levels, with the aim of optimizing a certain objective, such as minimizing the total operational cost.

Logistics optimization problems and their solutions have wide applications in many industries, such as in the shipping industry [25, 26, 30], the telecommunication industry [11, 19, 65], and the aviation industry [43, 49, 66], and have already

attracted great attention from modern companies around the world. The main reason is that, as businesses expand globally, and the scale of the decisions they face increases, more and more companies are turning to exploit the techniques in mathematics and computer science to model their practical operations into such problems and obtain solutions that assist them in making their decisions, thus further maintaining their competitive advantage.

Typically, solving such kinds of problems is a complex and challenging task faced by each operations manager, not only due to the scale and complexity of the problems, but also because most logistics optimization problems are known to be $\mathcal{NP}$-hard in theory, i.e., it is unlikely that any polynomial time algorithms exist that can solve them to optimality, which leads to them being intractable. Therefore, efficient methods of solving these problems are urgently needed, so as to find high quality solutions that can be used for decision support in everyday practice.

For certain logistics optimization problems, since it is not possible to determine optimal solutions in polynomial time through exact algorithms, a common solution method in practice is not to seek optimal solutions, but rather to seek near-optimal solutions in reasonable running time through developing efficient algorithms. In particular, the developed algorithms should have a "trade-off" between the running time and the solution quality. In the area of computer science, one of the typical methods of evaluating the "trade-off", which is also known as the performance of algorithms, is worst-case analysis, where polynomial time is used to measure the algorithm's running time, and an approximation ratio is used to measure the solution quality. For a minimization problem, an algorithm is said to have an approximation ratio ρ if it runs in polynomial time and can always produce a feasible solution with an objective value no more than ρ times that of an optimal solution, and hence the algorithm is defined as a ρ-approximation algorithm with constant ratio ρ. Moreover, the approximation ratio ρ is tight if there exists an instance of the problem to which

the solution produced by the algorithm is of an objective value exactly equal to ρ times that of an optimal solution [32]. Therefore, for certain logistics optimization problems that are $\mathcal{NP}$-hard, it is of great interest to develop efficient approximation algorithms with constant approximation ratios that produce near-optimal solutions in affordable running time.

In this thesis, we study two logistics optimization problems. The first problem, considered in Chapter 2, is called the *Capacitated Traveling Salesman Problem with Pickup and Delivery on a Tree*. The problem is a variant of the traveling salesman problem and makes operational level decisions. In particular, given a tree-shaped network and a vehicle with a finite capacity, the problem aims to determine the best route for this vehicle to transport amounts of a product from pickup points to delivery points on the network. The objective is to minimize the total travel distance of the vehicle. Assume that the given vehicle should start and end at a specified depot, and can transport any amount of the product collected from any pickup point to any delivery point, as only one type of product is involved. Moreover, the vehicle can serve each pickup or delivery point more than once, because the requests are splittable, but cannot temporarily unload and store any amount of the product prior to delivering it, because the delivery service is non-preemptive. In addition, we focus particularly on the case where pickup and delivery requests are balanced, due to the assumption that the depot can either absorb or supply any excess amount of the product.

This problem has several applications in logistics and transportation, because the tree-shaped network appears in many practical situations, including shorelines for cargo transportation, certain railway systems in pit mines, rural delivery systems with roads that branch off from a single highway, and certain inland water systems that include a main stream and several tributaries; for example, the systems in the Pearl River Delta of China is a typical tree-shaped network. More specifically, one possible application of the problem in such kind of water system is to decide on the

best route for a barge to take that repositions empty containers for shipping carriers, with the empty container that should be picked up or delivered at each port being decided by the existing and target amount of empty containers. To save on shipping cost, minimizing the total shipping distance during the repositioning is typically set as the optimization objective.

However, this problem is well-known to be strongly $\mathcal{NP}$-hard [74]. In the existing literature, we have found approximation algorithms for the problem on the general network, but not for the tree-shaped network we considered, and therefore the existing algorithms can be directly applied to the tree case. Nevertheless, the best existing approximation algorithm for the general case only has an approximation ratio of 5, and thus is expected to be further improved on in such a tree-shaped network. We therefore develop a 2-approximation algorithm for the problem, which is a significant improvement over the best constant approximation ratio of 5 derived from the existing literature. In particular, our algorithm extends an exact algorithm for the problem on a path developed by [74], where they first derived a lower bound on the number of traversals of each edge, and then constructed the route from a series of route lists. The route obtained is optimal because the number of traversals of each edge in it equals the lower bound. In order to obtain the route in the tree-shaped network, more complicated route lists have been considered, and we propose an algorithm to construct the route where the number of traversals of each edge is bounded by the lower bound in [74] plus two, and hence the new algorithm has an approximation ratio of 2. Moreover, we have conducted computational experiments over randomly generated instances, and the results show that our proposed algorithm also achieves good average performance, with a much shorter running time and better quality solution than a greedy algorithm.

The second problem, studied in Chapter 3, is called a *k-median Steiner forest problem*, which makes both strategic and tactical level decisions, including facility

locations and network connections. In particular, given that k is the maximum number of facilities that are allowed to be opened and a set of clients that need to be served through connecting to any open facility, the problem is to jointly optimize opening facilities and connect the open facilities to the clients by a path, such that the number of open facilities should not exceed the given k and each client must be connected. The objective is to minimize the total connection cost. Therefore, the resulting solution is a collection of at most k trees with a minimum total edge weight, where the trees should cover all the clients and each tree should contain a distinct facility as the root.

The problem has wide applications in both the transportation and telecommunication industries. In particular, carriers in the transportation industry often need to decide on the location of vehicle depots and how to connect these depots with the customers through road constructions. Moreover, this kind of application can be extended to more complicated cases, where plans of facility locations combined with vehicle routing or cargo shipping for carriers can be constructed. In the telecommunication industry, service providers often need to determine the best service center locations, as well as establish the optimum cable connections between the open centers and their clients, so as to provide high quality and effective services. In order to save on the total expense, their objective is to minimize the total used cable length, which is represented as the edge weight in the problem.

This problem, however, is also well-known to be strongly $\mathcal{NP}$-hard, because it contains the classical Steiner tree problem as a special case [50, 73]. In the literature, only a 2-approximation algorithm is known, this being based on a Lagrangian relaxation of the problem that relies on a sophisticated primal-dual schema. However, since the existing algorithm is extremely complicated, it is of great interest to develop simpler and improved algorithms for this problem. Therefore, we develop an improved approximation algorithm that includes only a simple transformation from

an optimal solution of a minimum spanning tree problem, and which has a polynomial time complexity. Compared with the existing algorithm, our new algorithm theoretically achieves a better and tight approximation ratio that is much easier to be proved. Computational results show that our algorithm also guarantees to produce solutions of equal or better quality over randomly generated instances, the improvement being, in some cases, up to 50%. Additionally, we have also considered two non-trivial special cases of this problem, where either every location contains a client, or all the locations are in a tree-shaped network. Both of these two cases are commonly seen in practice. For these two cases, whether there exist any algorithms that can solve them to optimality in polynomial time is still unknown. Therefore, we provide a positive answer to this question by proposing new algorithms to solve the above two special cases to optimality in polynomial time.

The remainder of this thesis is organized as follows. The above two problems, as well as the newly developed algorithms, will be presented in Chapter 2 and Chapter 3, respectively. In Chapter 4 we conclude this thesis, along with a discussion of future research directions for the two problems.

CHAPTER 2

An Improved Approximation Algorithm for the Capacitated TSP with Pickup and Delivery on a Tree

2.1 Introduction

The *Capacitated Traveling Salesman Problem with Pickup and Delivery* (CTSP-PD) is a variant of the Traveling Salesman Problem (TSP) [2, 37]. Consider a graph $G = (V, E)$, where V is the vertex set and E is the edge set. Each edge $e \in E$ is associated with a non-negative edge length denoted by $d(e)$. Each vertex $v \in V$ is associated with an amount of a product denoted by an integer $q(v)$. Each positive $q(v)$ indicates that v is a *pickup point* from which $q(v)$ units of the product need to be picked up, each negative $q(v)$ indicates that v is a *delivery point* to which $-q(v)$ units of the product need to be delivered, and each zero $q(v)$ indicates that v is a *transient point* having no requirement for pickups or deliveries. Consider a vehicle of capacity k that needs to start and end its route at a depot $s \in V$. Suppose that s can either absorb or supply any excess amount of the product so as to maintain a balance between pickups and deliveries, i.e., $\sum_{v \in V} q(v) = 0$. The goal of the CTSPPD is to minimize the length of the route for the vehicle to carry out all the pickup and de-

livery requests, but without ever exceeding its capacity. As only one type of product is involved, the vehicle can transport any amount of the product collected from any pickup point to any delivery point. Since pickup and delivery requests can both be split, the vehicle can serve each pickup or delivery point more than once. However, the delivery service is non-preemptive, and as a result the vehicle cannot temporarily unload and store any amount of the product prior to delivering it.

The CTSPPD has several applications in logistics [23, 74]. For example, heavy construction projects often require terrain modifications, which involve moving large volumes of earth from cut locations to fill locations by an earth-moving vehicle, where a cut location is a pickup point that supplies earth, and a fill location is a delivery point that needs earth [12, 35, 54]. To save the cost of earth excavation, the distance traveled by the vehicle needs to be minimized. Applications of the CTSPPD also occur in inventory repositioning, where retailers that have excess stock can serve as pickup points for supplying retailers that are short of stock [2]. Other applications of the CTSPPD with one pickup point include newspaper distribution [70], cattle feed distribution [58], and helicopter routing for crew changes [69].

This chapter studies the CTSPPD on a tree (CTSPPD-T), where the given graph is a tree, denoted by T, and the depot s is located at the root of T. Such tree-shaped networks appear in practical situations, including certain railway systems in pit mines [51], shorelines for cargo transportation [46], and rural delivery systems with roads that branch off from a single highway [13, 72]. In some water systems, such as those in the Pearl River Delta of China, the main stream and its tributaries also often form a tree-shaped network. One application of the CTSPPD-T in such a water system is to decide on the route for a barge that must reposition empty containers for shipping carriers, with the differences between existing and target number of empty containers at each port corresponding to the number that should be picked up or delivered.

The CTSPPD-T is strongly $\mathcal{NP}$-hard even when $q(v) = 1$ for all $v \in V$ [74].

Therefore, developing approximation algorithms that give near-optimal solutions is of great interest.

2.1.1 Literature Review

Approximation ratios known for the CTSPPD-T are all taken from the approximation algorithms for the CTSPPD. Chalasani and Motwani [18] first developed a $(5\alpha + 2 - 5\alpha/k)$-approximation algorithm for the CTSPPD, where α is the approximation ratio available for the TSP. Anily and Bramel [2] later devised two algorithms that improved the approximation ratio to $(4\alpha + 1 - 2\alpha/k)$, and $\alpha + \lfloor \log_2 k \rfloor/2 + (2\lfloor k/2 \rfloor - 1)/2^{\lceil \log_2 k \rceil}$. Since the TSP on a tree can be solved to optimality in polynomial time, one can apply the above three approximation algorithms to the CTSPPD-T as a special case to achieve approximation ratios of $(7 - 5/k)$, $(5 - 2/k)$, and $1 + \lfloor \log_2 k \rfloor/2 + (2\lfloor k/2 \rfloor - 1)/2^{\lceil \log_2 k \rceil}$. Thus, the existing best constant approximation ratio for the CTSPPD-T is 5.

It is important to note that all three constant ratio approximation algorithms reviewed above for the CTSPPD and the CTSPPD-T assume that each pickup or delivery point requests exactly one unit of the product. Although a pickup or delivery point requesting $q \geq 2$ units can be split into q identical points with each requesting one unit, the time complexity of the split is proportional to $\sum_{v \in V} |q(v)|$.

Moreover, for the CTSPPD with $k = 1$, several studies developed heuristics that exhibited good average performance over randomly generated instances but with no constant approximation ratio guarantees [35, 54]. For the CTSPPD-T with k equal to one or infinity, and for the CTSPPD on a path (CTSPPD-P), which is a special case of the CTSPPD-T with T being a single path, Wang et al. [74] showed that they can be solved to optimality in polynomial time.

The CTSPPD is different from other pickup and delivery problems [7, 48, 57, 59, 60, 68], such as the Swapping Problem and the TSP with Delivery and Backhauls.

In the Swapping Problem [3, 18, 68], products for pickup and delivery belong to multiple commodity types, and the set of products is partitioned into two subsets: preemptive products that can be stored temporarily at any intermediate vertex, and non-preemptive products that should be shipped directly from pickup points to delivery points. In the TSP with Delivery and Backhauls [4, 17], a vehicle needs to serve two types of demand, including the delivery demand that requests the vehicle to deliver products from the depot to a delivery point, and the backhaul demand that requests the vehicle to pick up products from a pickup point and return them to the depot.

Katoh and Yano [47] studied a variant of the TSP with Delivery and Backhauls, which, as with the CTSPPD-T, assumes that the given graph forms a tree, and that both delivery and backhaul demands can be split. Since the depot is the only destination of each backhaul demand and the only origin of each delivery demand, each sub-route of the vehicle between two consecutive visits to the depot can serve at most k delivery points and k pickup points. Based on this property, Katoh and Yano [47] developed a 2-approximation algorithm, and Asano et al. [9] showed that an approximation ratio of 1.351 can be achieved for a special case with no backhaul demand. However, in the CTSPPD-T, the property mentioned above does not hold, since in between two consecutive visits to the depot it is possible that the vehicle carries the product back and forth between pickup points and delivery points, and as a result serves more than k pickup points and k delivery points. Thus, the techniques in [9, 47] cannot be applied.

The CTSPPD is also a variant of the one commodity pickup and delivery TSP (1-PDTSP) [37, 39, 55]. Unlike the CTSPPD, the 1-PDTSP imposes a restriction such that the vehicle must follow a Hamiltonian tour so as to visit each vertex exactly once. Thus, the vehicle can pick up the product from each pickup point or deliver the product to each delivery point only once. Since for the 1-PDTSP it is strongly $\mathcal{NP}$-

hard to find even a feasible solution [37], existing works on the 1-PDTSP have focused on the development of exact algorithms for small-size instances [36, 38], or heuristics for large-size instances [37] with no constant approximation ratio guarantees.

2.1.2 Main Results

We develop a 2-approximation algorithm for the CTSPPD-T which improves on the best constant approximation ratio of 5 derived from the CTSPPD literature [2]. The proposed algorithm has a time complexity of $O(|V|(1 + \sum_{v \in V} |q(v)|/k))$, which is polynomial in the input size, as long as the ratio $(\sum_{v \in V} |q(v)|)/k$ is polynomially bounded by $|V|$. This improves on the time complexities of existing approximation algorithms derived from the CTSPPD literature [2, 18], which are polynomial in the input size only when $\sum_{v \in V} |q(v)|$ is polynomially bounded by $|V|$.

Our algorithm extends an exact algorithm for the CTSPPD-P developed by Wang et al. [74], who first derived a lower bound on the number of traversals of each edge, and then constructed a series of route lists to form an optimal route where the number of traversals of each edge equals the lower bound. For the CTSPPD-T, our construction of route lists is more complicated, and the route obtained from the route lists traverses each edge a number of times that is bounded from above by the lower bound plus two, leading to an approximation ratio of 2.

The remainder of this chapter is organized as follows. In Section 2.2, we formulate the problem and introduce notation. In Section 2.3, we define standard instances of the problem, and show that it is sufficient to consider only standard instances in this study. In Section 2.4, we define a structured route list with respect to each vertex of the given tree, and show that in order to achieve an approximation ratio of 2, it is sufficient to construct such a structured route list for the root. Based on this, we develop a 2-approximation algorithm in Section 2.5, and evaluate its performance over randomly generated instances in Section 2.6. The chapter is summarized in

Section 2.7.

2.2 Notation

Let $\mathcal{I} = (T, d, q, s, k)$ denote an instance of the CTSPPD-T, where tree $T = (V, E)$. For each $v \in V$, let $p(v)$ denote the parent of v in T, which is the vertex adjacent to v on the shortest path from v to the root s. Thus, v is called a child of $p(v)$. Let $e(v)$ denote the edge joining $p(v)$ and v. Let T_v denote the subtree of T rooted at v, and use $V(T_v)$ and $E(T_v)$ to denote the vertex and edge sets of T_v. Define the total balance of vertices in T_v as $q(T_v) := \sum_{v \in V(T_v)} q(v)$. For ease of presentation, let us assume $p(s) = s$, and assume $(v, v) \in E$ and $d(v, v) = 0$ for each $v \in V$. If a vertex v is not a leaf of T, we call it an *internal node* of T.

Along any route, the vehicle can either change its position from one vertex to the other vertex via an edge of T, or change its load by picking up or delivering a certain amount of the product from or to its current position. Define a state of the vehicle as a pair of its position and load. Thus, a route for the vehicle, denoted by σ, can be represented by a sequence of states $[\text{pos}(\sigma, i), \text{load}(\sigma, i)]$ for $1 \leq i \leq \text{size}(\sigma)$, where $\text{pos}(\sigma, i) \in V$ and $\text{load}(\sigma, i)$ with $0 \leq \text{load}(\sigma, i) \leq k$ indicate the position and the load of the vehicle in its i-th state, and $\text{size}(\sigma)$ indicates its total number of states. We refer to $[\text{pos}(\sigma, 1), \text{load}(\sigma, 1)]$ and $\text{load}(\sigma, 1)$ as the initial state and initial load of σ, and refer to $[\text{pos}(\sigma, \text{size}(\sigma)), \text{load}(\sigma, \text{size}(\sigma))]$ and $\text{load}(\sigma, \text{size}(\sigma))$ as the final state and final load of σ. Let $d(\sigma) := \sum_{i=2}^{\text{size}(\sigma)} d(\text{pos}(\sigma, i-1), \text{pos}(\sigma, i))$ represent the total length of σ. Hence, a *feasible route* σ for $\mathcal{I}$ is defined as a route that satisfies the following four conditions:

1. The vehicle starts at root s with an empty load, and also ends at s. That is, $\text{pos}(\sigma, 1) = \text{pos}(\sigma, \text{size}(\sigma)) = s$, and $\text{load}(\sigma, 1) = 0$.

2. The vehicle either moves through an edge of T, or stays at a vertex to pick

up or deliver the product. That is, for $2 \leq i \leq \text{size}(\sigma)$, either $(\text{pos}(\sigma, i-1), \text{pos}(\sigma, i)) \in E$ and $\text{load}(\sigma, i-1) = \text{load}(\sigma, i)$, or $\text{pos}(\sigma, i-1) = \text{pos}(\sigma, i)$ and $|\text{load}(\sigma, i) - \text{load}(\sigma, i-1)| \leq k$.

3. The vehicle satisfies all the pickup and delivery requests in T, picks up the product only from pickup points that still have supplies, and delivers the product only to delivery points that still have requirements. That is, along σ the vehicle picks up exactly $\max\{q(v), 0\}$ units of the product from v, and delivers exactly $\max\{-q(v), 0\}$ units to v, for each vertex $v \in V$.

4. The vehicle's load never falls below zero or exceeds capacity k. That is, $0 \leq \text{load}(\sigma, i) \leq k$ for $1 \leq i \leq \text{size}(\sigma)$.

The total length of an optimal route is indicated by $\text{OPT}(\mathcal{I})$. Although each edge $(u, v) \in E$ is undirected, a traversal of (u, v) can be in either direction, from u to v or from v to u, as represented by a traversal of $u \to v$ or a traversal of $v \to u$, respectively. Due to the balance between pickups and deliveries, every feasible route σ must end with an empty load, i.e., $\text{load}(\sigma, \text{size}(\sigma)) = 0$.

Moreover, we define a route list, denoted by $\vec{\sigma}$, as a sequence of routes σ_i for $1 \leq i \leq |\vec{\sigma}|$, where $|\vec{\sigma}|$ indicates the number of routes in $\vec{\sigma}$. We also define the total balance of routes in $\vec{\sigma}$ as their total final load minus their total initial load. Accordingly, if $\vec{\sigma}$ contains only one route σ, and σ is a feasible route, then the total balance of $\vec{\sigma}$ is zero.

2.3 Standard Instances

In this section, we first define standard instances of the CTSPPD-T in Definition 2.1, and then show in Theorem 2.1 that from any 2-approximation algorithm for standard instances we can obtain a 2-approximation algorithm for arbitrary instances.

Hence, it is sufficient to consider only standard instances throughout the remainder of this chapter.

Definition 2.1. A CTSPPD-T instance (T, d, q, s, k) is standard if, and only if, it satisfies:

1. Each leaf of T is either a pickup point, a delivery point, or the root s;
2. Each internal node of T is a transient point;
3. T is a full binary tree, i.e., a tree in which every internal node has exactly two children;
4. For each internal node v of T, the left child l and the right child r of v satisfy $q(T_l) \geq q(T_r)$.

According to Definition 2.1, a tree T of a standard instance is a full binary tree in which each internal node v satisfies $q(v) = 0$ and $q(T_l) \geq q(T_r)$ for its left child l and right child r. Thus, when constructing a partial solution that serves requests in subtree T_v, we need to handle only these three cases: (i) with $q(T_l) \geq 0$ and $q(T_r) \geq 0$, or $q(T_l) < 0$ and $q(T_r) < 0$; (ii) with $q(T_l) \geq 0$, $q(T_r) < 0$, and $q(T_v) \geq 0$; and (iii) with $q(T_l) \geq 0$, $q(T_r) < 0$, and $q(T_v) < 0$, respectively. Compared with having to take into account all problem instances, the number of cases to be handled for only standard instances is much smaller. As a result, the presentation of our approximation algorithm, as shown later on in Section 2.5, can be much simpler.

To establish Theorem 2.1, we define traversal (σ, e) as the number of traversals of edge e by σ (in either direction) for each edge $e \in E$.

Theorem 2.1. *Each CTSPPD-T instance $\mathcal{I}' = (T', d', q', s', k')$ with $T' = (V', E')$ can be transformed in $O(|V'|)$ time to a standard instance $\mathcal{I} = (T, d, q, s, k)$ with $T = (V, E)$, such that each feasible route σ of $\mathcal{I}$ can be transformed to a feasible route σ' of $\mathcal{I}'$ in $O(|V'| \max_{e \in E} \text{traversal}(\sigma, e))$ time with $d'(\sigma')/\text{OPT}(\mathcal{I}') \leq d(\sigma)/\text{OPT}(\mathcal{I})$.*

Proof. Consider Definition 2.1. Condition 1 can be assumed for $\mathcal{I}'$ without loss of generality, since each leaf that is a transient point but not the root can be removed. If Condition 3 is satisfied, then Condition 4 can be assumed without loss of generality. Thus, we need to consider only the case when $\mathcal{I}'$ does not satisfy either Condition 2 or Condition 3.

If $\mathcal{I}'$ does not satisfy Condition 2, there exists an internal node v of T' with $|q'(v)| > 0$. To transform $\mathcal{I}'$ to an instance $\mathcal{I}$ that satisfies Condition 2, we can initialize $\mathcal{I} \leftarrow \mathcal{I}'$, and then revise $\mathcal{I}$, by inserting a new child u to v with $d(v,u) \leftarrow 0$ and $q(u) \leftarrow q'(v)$, and then setting $q(v) \leftarrow 0$. By doing so, the optimal route length is not changed, and so we can replace each pickup or delivery at u in a feasible route σ of $\mathcal{I}$ with a pickup or delivery at v, thus obtaining a feasible route σ' of $\mathcal{I}'$, but without changing the route length. Thus, $d'(\sigma')/\mathrm{OPT}(\mathcal{I}') \leq d(\sigma)/\mathrm{OPT}(\mathcal{I})$. By setting $\mathcal{I}' \leftarrow \mathcal{I}$, the above transformation can be iterated until $\mathcal{I}$ satisfies Condition 2.

Assume that $\mathcal{I}'$ now satisfies Condition 2. Next, we show that $\mathcal{I}'$ can be transformed to an instance $\mathcal{I}$ where each internal node has at least two children. Consider any internal node v of T' that has exactly one child u. According to Condition 2, v must be a transient node. Initialize $\mathcal{I} \leftarrow \mathcal{I}'$, and then consider the following two cases. Case 1: v is not the root. We can revise $\mathcal{I}$ without changing the optimal route length by replacing edges $(p(v), v)$ and (v, u) with $(p(v), u)$, setting $d(p(v), u) \leftarrow d'(p(v), v) + d'(v, u)$, and then removing v. We can thus replace each traversal of $p(v) \rightarrow u$ or $u \rightarrow p(v)$ in a feasible route σ of $\mathcal{I}$ with $p(v) \rightarrow v \rightarrow u$ or $u \rightarrow v \rightarrow p(v)$, thereby constructing a feasible route σ' of $\mathcal{I}'$, but without changing the route length. Thus, $d'(\sigma')/\mathrm{OPT}(\mathcal{I}') \leq d(\sigma)/\mathrm{OPT}(\mathcal{I})$. Case 2: v is the root. According to Condition 1, a feasible route of $\mathcal{I}'$ must traverse $v \rightarrow u$ and $u \rightarrow v$ at least once each way. Thus, we can revise $\mathcal{I}$ by removing (v, u) and setting u as the root, which decreases the optimal route length by at least $2d'(v, u)$. Accordingly, any feasible route σ of $\mathcal{I}$ can be transformed to a feasible route σ'

of $\mathcal{I}'$ by adding a traversal of $v \to u$ at the beginning of σ, and a traversal of $u \to v$ at the end of σ, which increases the route length by $2d'(v,u)$. Thus, since $\mathrm{OPT}(\mathcal{I}) \leq \mathrm{OPT}(\mathcal{I}') - 2d'(v,u)$, $d'(\sigma') = d(\sigma) + 2d'(v,u)$ and $d(\sigma) \geq \mathrm{OPT}(\mathcal{I})$, it can be seen that $d'(\sigma')/\mathrm{OPT}(\mathcal{I}') \leq d(\sigma)/\mathrm{OPT}(\mathcal{I})$. By setting $\mathcal{I}' \leftarrow \mathcal{I}$, the above transformation can be iterated until each internal node has at least two children.

Now assume that each internal node of T' has at least two children. Thus, if $\mathcal{I}'$ does not satisfy Condition 3 of Definition 2.1, there must exist an internal node v of T' with at least three children, denoted by $u_1, ..., u_t$ for $t \geq 3$. To transform $\mathcal{I}'$ to an instance $\mathcal{I}$ that satisfies Condition 3, initialize $\mathcal{I} \leftarrow \mathcal{I}'$, and revise $\mathcal{I}$, by choosing u_1 as the left child of v, adding a new transient point u as the right child of v with $d(u,v) \leftarrow 0$, and moving each u_j for $2 \leq j \leq t$ to be a child of u with $d(u,u_j) \leftarrow d'(v,u_j)$. By doing so, the optimal route length is not changed, and by replacing each visit to u in a feasible route σ of $\mathcal{I}$ with a visit to v, we can obtain a feasible route σ' of $\mathcal{I}'$ without changing the route length. Thus, $d'(\sigma')/\mathrm{OPT}(\mathcal{I}') \leq d(\sigma)/\mathrm{OPT}(\mathcal{I})$. By setting $\mathcal{I}' \leftarrow \mathcal{I}$, the above transformation can be iterated until $\mathcal{I}$ satisfies Condition 3.

Hence, $\mathcal{I}'$ can be transformed to a standard instance $\mathcal{I}$. The transformation takes only $O(|V'|)$ time, since $|E'| \leq |V'| - 1$, each part of the transformation must stop after $O(|V'| + |E'|)$ iterations, and each iteration takes $O(1)$ time. This implies that V contains at most $O(|V'|)$ vertices. As shown above, each feasible route σ of $\mathcal{I}$ can be transformed backward to a feasible route σ' of $\mathcal{I}'$. Thus, the transformation from σ to σ' takes $O(|V'| \max_{e \in E} \mathrm{traversal}(\sigma, e))$ time, since (i) V contains at most $O(|V'|)$ vertices; (ii) in the transformation, at most $O(|V| \max_{e \in E} \mathrm{traversal}(\sigma, e))$ new states are created; and (iii) each deletion or replacement of a state takes $O(1)$ time. □

2.4 Structured Route Lists

Consider any standard instance $\mathcal{I}$. In this section, we introduce a structured route list, called a $\langle v, w \rangle$-route list, for each vertex $v \in V$, where w with $0 \leq w \leq k$ is an

integer parameter indicating the initial load of the first route in the route list. The 2-approximation algorithm developed in Section 2.5 is based on the construction of an $\langle s, 0\rangle$-route list.

Before introducing the definition of the $\langle v, w\rangle$-route list, let us make the following three observations from any feasible route σ and vertex $v \in V$:

Observation 1. Let $\vec{\pi}(\sigma, v)$ indicate a list of sub-routes of σ, where $\pi_i(\sigma, v)$ for $1 \leq i \leq |\vec{\pi}(\sigma, v)|$ represents the sub-route between the i-th traversal of $p(v) \to v$ and the i-th traversal of $v \to p(v)$ in σ. Thus, each $\pi_i(\sigma, v)$ starts and ends at $p(v)$, visiting only vertices of T_v in between, and $|\vec{\pi}(\sigma, v)|$ equals both the total number of traversals of $p(v) \to v$ and the total number of traversals of $v \to p(v)$. To fulfill the pickup and delivery requests in T_v, the total balance of routes in $\vec{\pi}(\sigma, v)$ must equal $q(T_v)$. Since $\mathcal{I}$ is standard and the vehicle has a capacity of k, edge $e(v)$ must be traversed at least $2\max\{\lceil |q(T_v)|/k \rceil, 1\}$ times. Thus, define

$$n(v) := \max\{\lceil |q(T_v)|/k \rceil, 1\}. \tag{2.1}$$

We obtain $|\vec{\pi}(\sigma, v)| \geq n(v)$, and a lower bound on $\mathrm{OPT}(\mathcal{I})$ as shown below:

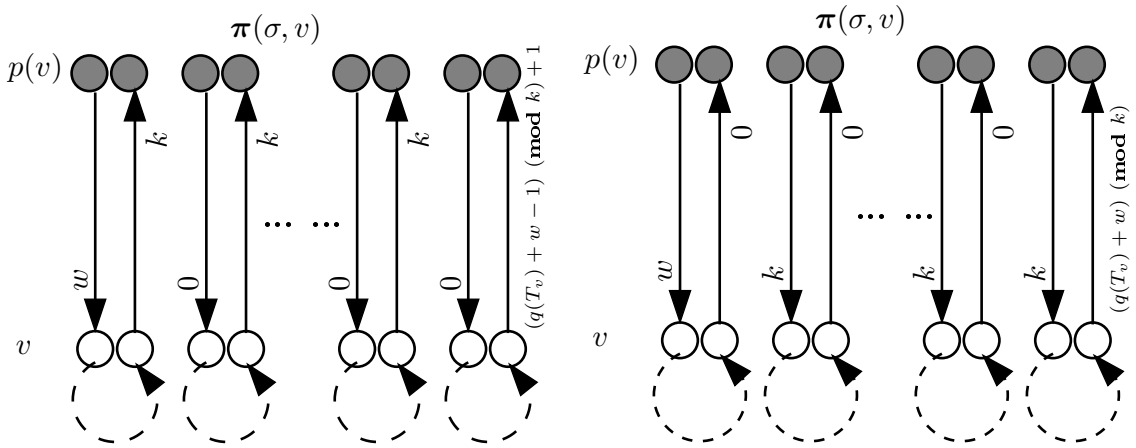
$$\sum_{u \in V} 2n(u)d(e(u)) \leq \mathrm{OPT}(\mathcal{I}). \tag{2.2}$$

Observation 2. By (2.1) and (2.2), if $|\vec{\pi}(\sigma, u)| \leq n(u) + 1$ for all $u \in V$, then $d(\sigma) \leq 2\mathrm{OPT}(\mathcal{I})$. Assuming $|\vec{\pi}(\sigma, u)| \leq n(u) + 1$ for $u \in V(T_v) \setminus \{v\}$, let us consider how to ensure $|\vec{\pi}(\sigma, v)| \leq n(v)+1$. One possible way is to require that $\vec{\pi}(\sigma, v)$ complies with a *greedy structure*, as defined in the following two cases, so that routes in $\vec{\pi}(\sigma, v)$ achieve a total balance of $q(T_v)$ with each route carrying as much of the product as possible out of or into T_v, given that the first route in $\vec{\pi}(\sigma, v)$ is forced to have an initial load of w, where $0 \leq w \leq k$:

Case 1: $q(T_v) \geq 0$. The greedy structure requires that the route list consists of a minimum number of routes to carry $q(T_v)$ more units of the product out of T_v than into T_v, such that: (i) each route, other than the last one, must carry a full final load of k units out of T_v; (ii) each route, other than the first one, must carry an empty initial load into T_v; and (iii) the final load of the last route cannot be empty if $q(T_v)+w > 0$. Thus, if $q(T_v) = 0$ and $w = 0$, then the route list consists of only one route with empty initial and final loads. If not, then $q(T_v) + w > 0$, and as shown in Figure 2.1(a), the route list consists of $\lceil (q(T_v) + w)/k \rceil$ routes, with the last route having a final load of $(q(T_v)+w-1) \pmod k + 1$, which equals k if $q(T_v)+w > 0$ and $(q(T_v)+w) \pmod k = 0$, and equals $(q(T_v)+w) \pmod k$ otherwise.

Case 2: $q(T_v) < 0$. The greedy structure requires that the route list consists of a minimum number of routes to carry $-q(T_v)$ more units of the product into T_v than out of T_v, such that: (i) each route, other than the first one, must carry a full initial load of k units into T_v; and (ii) each route, other than the last one, must carry an empty final load out of T_v. Accordingly, as shown in Figure 2.1(b), the route list consists of $\lceil -(q(T_v) + w)/k \rceil + 1$ routes, with the last route having a final load of $(q(T_v) + w) \pmod k$.

Let $m(v, w)$ and $\text{tail}(v, w)$ denote the number of routes, and the load of the final one, in a route list that complies with the greedy structure with respect to v and w.



(a) $q(T_v) \geq 0$ and $q(T_v) + w > 0$ (b) $q(T_v) < 0$

Figure 2.1: Illustration of the greedy structure.

According to the arguments above, we have that

$$m(v,w) = \begin{cases} 1, & \text{when } q(T_v) = 0 \text{ and } q(T_v) + w = 0, \\ \lceil -(q(T_v) + w)/k \rceil + 1, & \text{when } q(T_v) < 0, \\ \lceil (q(T_v) + w)/k \rceil, & \text{when } q(T_v) \geq 0 \text{ and } q(T_v) + w > 0. \end{cases} \tag{2.3}$$

$$\text{tail}(v,w) = \begin{cases} 0, & \text{when } q(T_v) = 0 \text{ and } q(T_v) + w = 0, \\ (q(T_v) + w) \pmod{k}, & \text{when } q(T_v) < 0, \\ (q(T_v) + w - 1) \pmod{k} + 1, & \text{when } q(T_v) \geq 0 \text{ and } q(T_v) + w > 0. \end{cases} \tag{2.4}$$

Thus, from (2.1), (2.3), and $0 \leq w \leq k$, we obtain that

$$n(v) \leq m(v,w) \leq n(v) + 1. \tag{2.5}$$

Furthermore, we can now establish Lemma 2.1, which provides unified representations of $m(v,w)$ and $\text{tail}(v,w)$ for the cases when $q(T_v) + w > 0$ and when $q(T_v) + w < k$, respectively.

Lemma 2.1. *Consider any integer w with $0 \leq w \leq k$.*

1. *If $q(T_v) + w > 0$, then $m(v,w) = \lceil (q(T_v) + w)/k \rceil$ and $\text{tail}(v,w) = (q(T_v) + w -$*

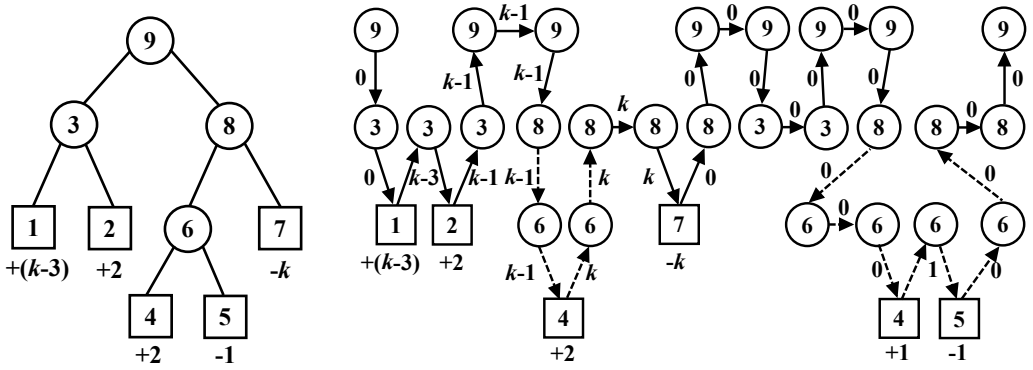
1) (mod k) + 1*;*

2. *If* $q(T_v) + w < k$, *then* $m(v, w) = \lceil -(q(T_v) + w)/k \rceil + 1$ *and* tail$(v, w) = (q(T_v) + w)$ (mod k)*.*

Proof. To prove Item 1 in Lemma 2.1, consider the case when $q(T_v) + w > 0$. If $q(T_v) \geq 0$, then due to (2.3) and (2.4), Item 1 is true. Otherwise, $q(T_v) < 0$, which implies that: (i) $m(v, w) = \lceil -(q(T_v) + w)/k \rceil + 1$ due to (2.3); (ii) tail$(m, w) = (q(T_v) + w)$ (mod k) due to (2.4); and (iii) $0 < q(T_v) + w < k$. Moreover, since $0 < q(T_v) + w < k$, we have $1 = \lceil (q(T_v) + w)/k \rceil = \lceil -(q(T_v) + w)/k \rceil + 1$, and $(q(T_v) + w - 1)$ (mod k) $+ 1 = (q(T_v) + w)$ (mod k). Thus, Item 1 is true.

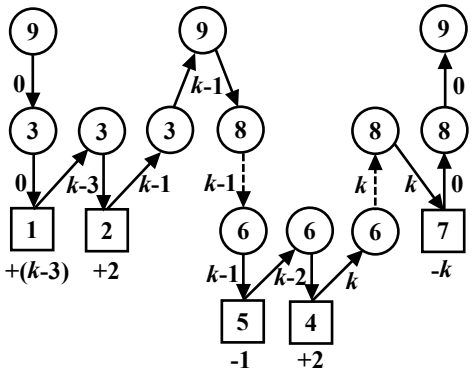
To prove Item 2 in Lemma 2.1, consider the case when $q(T_v) + w < k$. If $q(T_v) < 0$, then due to (2.3) and (2.4), Item 2 is true. Otherwise, $q(T_v) \geq 0$, which implies that $0 \leq q(T_v) + w < k$. If $q(T_v) + w = 0$, then $q(T_v) = w = 0$, which, taken together with (2.3), implies that $m(v, w) = 1 = \lceil -(q(T_v) + w)/k \rceil + 1$, and, taken together with (2.4), implies that tail$(v, w) = 0 = (q(T_v) + w)$ (mod k). Thus, Item 2 is true. Otherwise, $0 < q(T_v) + w < k$, which, taken together with (2.3), implies that $m(v, w) = \lceil (q(T_v) + w)/k \rceil = 1 = \lceil -(q(T_v) + w)/k \rceil + 1$, and, taken together with (2.4), implies that tail$(m, w) = (q(T_v) + w - 1)$ (mod k) $+ 1 = (q(T_v) + w)$ (mod k). Thus, Item 2 is true. □

Observation 3. When $m(v, w) = n(v)$, we can observe from (2.5) that in order to ensure $|\vec{\pi}(\sigma, v)| \leq n(v) + 1$ it is not necessary for $\vec{\pi}(\sigma, v)$ to consist of only $m(v, w)$ routes that comply with the greedy structure. Instead, $\vec{\pi}(\sigma, v)$ can contain an additional route that has empty initial and final loads, leading to $|\vec{\pi}(\sigma, v)| = n(v) + 1$. Moreover, this additional route might not be an empty route, as it can be used to transport the product within T_v. For example, consider the standard instance $\mathcal{I}$ in Figure 2.2(a), where $n(u) = 1$ for all $u \in V$ by (2.1). Consider the feasible route σ shown in Figure 2.2(b), where the two sub-routes shown by dotted lines form $\vec{\pi}(\sigma, 6)$



(a) A standard instance with capacity $k \geq 4$.

(b) A feasible route σ, where integers indicate load changes of the vehicle, and the sub-routes shown by dotted lines form $\vec{\pi}(\sigma, 6)$.



(c) An optimal route σ^*, where integers indicate the load changes of the vehicle.

Figure 2.2: Illustration of observations on $\vec{\pi}(\sigma, v)$, where circles indicate internal nodes, rectangles indicate leaves, numbers inside the circles and rectangles indicate vertex indices, numbers below the rectangles indicate product amounts, arrows indicate the vehicles' routes, numbers along the arrows indicate the vehicle's loads, and vertex 9 is the root of the tree.

with $w = k - 1$. By (2.1) and (2.3), $m(6, w) = 1 = n(6)$. It can be seen that the first route in $\vec{\pi}(\sigma, 6)$ complies with the greedy structure in picking up one unit from vertex 4 and carrying it out of T_6. The second route in $\vec{\pi}(\sigma, 6)$, which carries empty initial and final loads, only transports one unit from vertex 4 to vertex 5 inside T_6. Thus, σ traverses edge $(6, 8)$ four times. Since another feasible route σ^* shown in Figure 2.2(c) traverses each edge twice, we obtain that σ^* is an optimal route, but σ is not.

Based on the above three observations of $\vec{\pi}(\sigma, v)$, we define the $\langle v, w \rangle$-route list as

follows.

Definition 2.2. For any vertex $v \in V$ and any integer w with $0 \leq w \leq k$, a route list $\vec{\sigma}$ is a $\langle v, w \rangle$-route list if, and only if, $\vec{\sigma}$ satisfies the following conditions:

1. Each route starts and ends at $p(v)$, visiting only vertices of T_v in between;
2. Routes in $\vec{\sigma}$ serve all the pickup and delivery requests in T_v;
3. Routes in $\vec{\sigma}$ traverse each $e(u)$ with $u \in V(T_v) \setminus \{v\}$ at most $2n(u) + 2$ times in total;
4. $\vec{\sigma}$ consists of exactly $n(v) + 1$ routes, denoted by σ_i for $1 \leq i \leq n(v) + 1$;
5. The initial load of the first route σ_1 equals w;
6. The first $m(v, w)$ routes, σ_i for $1 \leq i \leq m(v, w)$, comply with the greedy structure;
7. If $m(v, w) = n(v)$, then $\sigma_{n(v)+1}$ has an empty initial load and an empty final load.

Thus, we can establish Theorem 2.2, which implies that in order to develop a 2-approximation algorithm for the CTSPPD-T, it is sufficient to construct an $\langle s, 0 \rangle$-route list.

Theorem 2.2. *Given any $\langle s, 0 \rangle$-route list, it takes $O(1)$ time to construct a feasible route with a total length not greater than* $2\mathrm{OPT}(\mathcal{I})$.

Proof. Consider any $\langle s, 0 \rangle$-route list $\vec{\sigma}_s$. Since $q(T_s) = 0$, we have $n(s) + 1 = 2$ by (2.1), $m(s, 0) = 1$ by (2.3), and $\mathrm{tail}(s, 0) = 0$ by (2.4). Thus, by Conditions 2 and 4 of Definition 2.2, $\vec{\sigma}_s$ contains exactly two routes, denoted by $\sigma_{s,1}$ and $\sigma_{s,2}$, which together serve all the pickup and delivery requests in T. By Conditions 3 and 4 of Definition 2.2, $\sigma_{s,1}$ and $\sigma_{s,2}$ together traverse each $e(u)$ with $u \in V$ at most $(2n(u)+2)$

times, which, together with (2.2), implies that $d(\sigma_{s,1}) + d(\sigma_{s,2}) \leq 2\text{OPT}(\mathcal{I})$. Since $m(s,0) = 1 = n(s)$ and $\text{tail}(s,0) = 0$, by Conditions 5–7 of Definition 2.2, both $\sigma_{s,1}$ and $\sigma_{s,2}$ have empty initial and final loads. Thus, they can be linked in $O(1)$ time to form a feasible route that follows $\sigma_{s,1}$ and $\sigma_{s,2}$ sequentially, with a total length not greater than $2\text{OPT}(\mathcal{I})$. □

2.5 The 2-Approximation Algorithm

Consider any standard instance $\mathcal{I}$. In this section, we follow the implication of Theorem 2.2 to develop a 2-approximation algorithm for the CTSPPD-T, by constructing an $\langle s, 0\rangle$-route list.

Without loss of generality, suppose that vertices in V are re-labeled by $1, 2, ..., |V|$ according to the postorder traversal sequence of T [28], so that each vertex is labeled after all the vertices in its subtrees are labeled. As a result, we have $s = |V|$ and $p(v) > v$ for each $v \in V \setminus \{s\}$. Our construction of an $\langle s, 0\rangle$-route list uses a recursive process, which constructs a $\langle v, w_v\rangle$-route list for $v = 1, 2, ..., |V| - 1, |V|$. Here, w_v represents the value of the initial load of the first route in the route list to be constructed for each vertex v, and its value can be different for different v. To ensure that the recursive construction of the route lists is feasible, we will fix $w_s = 0$, and predetermine values of w_v for other vertices v in Section 2.5.1. Based on several operators defined in Section 2.5.2 on routes and route lists, we will first present in Section 2.5.3 a sub-routine to construct a $\langle v, w_v\rangle$-route list for each leaf v, and then, for each internal node v, given an $\langle l, w_l\rangle$-route list $\vec{\sigma}_l$ and an $\langle r, w_r\rangle$-route list $\vec{\sigma}_r$ for the left child l and the right child r of v, respectively, we will present in Section 2.5.4 three sub-routines that can be used recursively to construct a $\langle v, w_v\rangle$-route list from $\vec{\sigma}_l$ and $\vec{\sigma}_r$. Based on these constructions, we can develop the main algorithm and prove its approximation ratio in Section 3.2.1.

2.5.1 Computation of w_v

Before constructing the $\langle v, w_v \rangle$-route lists, we need to first determine the value of each w_v, using the following iterative process:

1. Initialize $w_s \leftarrow 0$, since $s = |V|$ is the root.

2. For each $v = |V|, |V|-1, ..., 1$, if v is an internal node, whose left and right children are denoted by $l < v$ and $r < v$, then set $w_l \leftarrow w_v$ and $w_r \leftarrow \text{tail}(l, w_l)$.

It can be seen that the above process starts with $w_s = 0$, runs in $O(|V|)$ time, and ends with values of w_v satisfying $0 \leq w_v \leq k$ for $v \in V$. By following this process, we can compute the values of w_v for the instance in Figure 2.2(a), these being shown in Table 2.1 along with the values of $m(v, w_v)$, $\text{tail}(v, w_v)$, and $n(v)$.

Table 2.1: Computation of w_v, $m(v, w_v)$, $\text{tail}(v, w_v)$, and $n(v)$ for the instance in Figure 2.2(a).

v	1	2	3	4	5	6	7	8	9
$q(v)$	$k-3$	2	0	2	-1	0	$-k$	0	0
$q(T_v)$	$k-3$	2	$k-1$	2	-1	1	$-k$	$1-k$	0
w_v	0	$k-3$	0	$k-1$	1	$k-1$	k	$k-1$	0
$q(T_v)+w_v$	$k-3$	$k-1$	$k-1$	$k+1$	0	k	0	0	0
$m(v, w_v)$	1	1	1	2	1	1	1	1	1
$\text{tail}(v, w_v)$	$k-3$	$k-1$	$k-1$	1	0	k	0	0	0
$n(v)$	1	1	1	1	1	1	1	1	1

For any internal node v with the left and right children denoted by l and r, consider any $\langle l, w_l \rangle$-route list $\vec{\sigma}_l$ and $\langle r, w_r \rangle$-route list $\vec{\sigma}_r$. Due to Step 2 above, we have $w_r = \text{tail}(l, w_l)$. For example, Table 2.1 shows that $w_2 = \text{tail}(1, w_1) = k-3$ for the instance in Figure 2.2(a). This, together with Condition 6 of Definition 2.2, implies that the final load of the $m(l, w_l)$-th route in $\vec{\sigma}_l$ always equals the initial load of the first route in $\vec{\sigma}_r$. Thus, these two routes can be linked together to form a new route. This property has been used extensively later on in Section 2.5.4 for the construction of a $\langle v, w_v \rangle$-route list from $\vec{\sigma}_l$ and $\vec{\sigma}_r$.

2.5.2 Operators on Routes and Route Lists

To make it easier when presenting our construction of the $\langle v, w_v \rangle$-route lists, we here define the following operators:

1. Add($\vec{\sigma}, \sigma$): Given a route list $\vec{\sigma}$ and a route σ, Add($\vec{\sigma}, \sigma$) adds σ at the end of $\vec{\sigma}$.

2. Extract($\vec{\sigma}, i_1, i_2$): Given a route list $\vec{\sigma}$, and two integers i_1 and i_2, where $1 \leq i_1, i_2 \leq |\vec{\sigma}|$, Extract($\vec{\sigma}, i_1, i_2$) returns a route list that consists of the routes σ_i in $\vec{\sigma}$ for each $i_1 \leq i \leq i_2$ if $i_1 \leq i_2$, and returns an empty route list otherwise.

3. Extend($\vec{\sigma}, v$): Given a vertex $v \in V$ and a route list $\vec{\sigma}$ with each route starting and ending at v, Extend($\vec{\sigma}, v$) extends each route in $\vec{\sigma}$ to start and end at $p(v)$. Specifically, each route σ in $\vec{\sigma}$ is considered. If σ is not empty, then Extend($\vec{\sigma}, v$) replaces σ with a route that starts from $p(v)$ with an initial load equal to load($\sigma, 0$), then traverses $p(v) \rightarrow v$, follows the states of σ, and then traverses $v \rightarrow p(v)$; otherwise, σ must be empty, and Extend($\vec{\sigma}, v$) replaces σ with a degenerate route that consists of only $p(v)$ with an empty load.

4. Link(σ, π): Given two routes, σ and π, such that either σ or π is empty, or that the final state of σ equals the initial state of π, Link(σ, π) returns a route that follows σ and then follows π.

5. CrossLink($\vec{\sigma}, \vec{\pi}$): Given two route lists, $\vec{\sigma}$ and $\vec{\pi}$, containing the same number of routes h, such that the final state of route σ_i equals the initial state of route π_i for $1 \leq i \leq h$, and that the final state of route π_i equals the initial state of route σ_{i+1}, for $1 \leq i \leq h-1$, CrossLink($\vec{\sigma}, \vec{\pi}$) returns a route that follows σ_1, π_1, σ_2, π_2, ..., σ_h, and π_h, sequentially.

6. Replace($\vec{\sigma}, i, \pi$): Given a route list $\vec{\sigma}$, an integer i with $1 \leq i \leq |\vec{\sigma}|$, and a route π, Replace($\vec{\sigma}, i, \pi$) replaces route σ_i with route π in $\vec{\sigma}$.

By storing both the states for routes, and the routes for route lists, as linked lists, it is easy to see that the following time complexities can be achieved for the above operators.

Lemma 2.2. *If linked lists are used as the data structure for routes and route lists, then*

1. $\text{Add}(\tilde{\sigma}, \sigma)$ *and* $\text{Link}(\sigma, \pi)$ *run in* $O(1)$ *time;*

2. $\text{Extract}(\vec{\sigma}, i_1, i_2)$, $\text{Extend}(\vec{\sigma}, v)$, *and* $\text{Replace}(\vec{\sigma}, i, \sigma)$ *run in* $O(|\vec{\sigma}|)$ *time;*

3. $\text{CrossLink}(\vec{\sigma}, \vec{\pi})$ *runs in* $O(|\vec{\sigma}| + |\vec{\pi}|)$ *time.*

2.5.3 Construction of $\langle v, w_v \rangle$-Route Lists for Leaves

The construction of $\langle v, w_v \rangle$-route lists for each leaf v of T is the basis of our approximation algorithm. To understand the basic concept, consider leaf 2 and leaf 4 of the instance in Figure 2.2(a). According to Table 2.1, $w_2 = k - 3$ and $w_4 = k - 1$, which implies that $q(2) + w_2 = k - 1$ and $q(4) + w_4 = k + 1$. We can construct a $\langle 4, w_4 \rangle$-route list $\vec{\sigma}_4$ by following the greedy structure, as explained in Section 2.4, to carry as much of the product as possible out of T_4, which, due to the capacity limit of k, needs to contain two routes, with the first carrying a full final load and the second carrying a final load of 1 unit, as shown in Figure 2.3. Similarly, we can construct a $\langle 2, w_2 \rangle$-route list $\vec{\sigma}_2$, which, due to $0 \leq q(2) + w_2 < k$, needs only the first route to follow the greedy structure and pick up both of the two units of the product out of T_2, as shown in Figure 2.3. Since $m(2, w_2) = 1 = n(2)$, $\vec{\sigma}_2$ needs to contain a degenerate route that consists of only vertex 3 with an empty load, in order that $\vec{\sigma}_2$ can satisfy Condition 7 of Definition 2.2 as a $\langle 2, w_2 \rangle$-route list.

We can now generalize the idea above so as to construct a $\langle v, w_v \rangle$-route list $\vec{\sigma}_v$ for each leaf v, using the following three stages.

Stage 1 (Initialization): Set $\vec{\sigma}_v \leftarrow \emptyset$.

Stage 2 (Construction): We first follow the greedy structure, as defined in Section 2.4, to construct routes $\sigma_{v,i}$ for $1 \leq i \leq m(v, w_v)$. Consider the following three cases:

- Case 1: $0 \leq q(v) + w_v \leq k$. Thus, $m(v, w_v) = 1$ by (2.3), and $\text{tail}(v, w_v) = q(v) + w_v$ by (2.4). To comply with the greedy structure, only one route $\sigma_{v,1}$ is needed, set as $[p(v), w_v][v, w_v][v, w_v + q(v)][p(v), w_v + q(v)]$, carrying an initial load of w_v units of the product, and then picking up $q(v)$ units from v (if $q(v) \geq 0$), or delivering $-q(v)$ units to v (if $q(v) < 0$).

- Case 2: $q(v) + w_v > k$. Thus, $m(v, w_v) \geq 2$ by (2.3), and $q(v) > 0$ by $w_v \leq k$. To comply with the greedy structure, each route needs to pick up as much of the product as possible from v. Thus, set $\sigma_{v,1} \leftarrow [p(v), w_v][v, w_v][v, k][p(v), k]$ as the first route, carrying an initial load of w_v units and picking up $(k - w_v)$ units from v. If $m(v, w_v) \geq 3$, set $\sigma_{v,i} \leftarrow [p(v), 0][v, 0][v, k][p(v), k]$ for $2 \leq i \leq m(v, w_v) - 1$, carrying an empty initial load and picking up k units from v. Finally, set $\sigma_{v,m(v,w_v)} \leftarrow [p(v), 0][v, 0][v, \text{tail}(v, w_v)][p(v), \text{tail}(v, w_v)]$ as the last route, carrying an empty initial load and picking up $\text{tail}(v, w_v)$ units from v to clear the remaining supplies.

- Case 3: $q(v) + w_v < 0$. Thus, $m(v, w_v) \geq 2$ by (2.3), and $q(v) < 0$ by

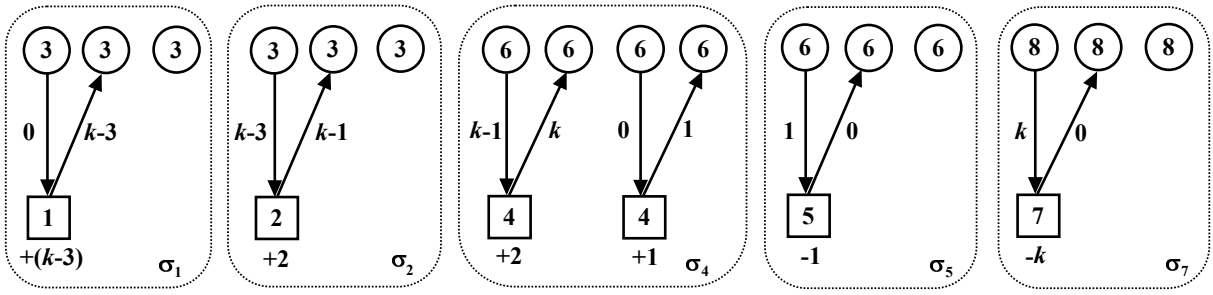


Figure 2.3: Construction of $\langle v, w_v \rangle$-route lists $\vec{\sigma}_v$ for leaves $v \in \{1, 2, 4, 5, 7\}$ of the instance in Figure 2.2(a).

$w_v \geq 0$. To comply with the greedy structure, each route needs to deliver as much of the product as possible to v. Similar to Case 2, set $\sigma_{v,1} \leftarrow [p(v), w_v][v, w_v][v, 0][p(v), 0]$ as the first route, carrying an initial load of w_v units and delivering it all to v. If $m(v, w_v) \geq 3$, set $\sigma_{v,i} \leftarrow [p(v), k][v, k][v, 0][p(v), 0]$ for $2 \leq i \leq m(v, w_v) - 1$, carrying an initial load of k units and delivering it all to v. Finally, set $\sigma_{v,m(v,w_v)} \leftarrow [p(v), k][v, k][v, \text{tail}(v, w_v)][p(v), \text{tail}(v, w_v)]$ as the last route, carrying an initial load of k units and delivering $k - \text{tail}(v, w_v)$ units to v to meet the remaining requirements.

We then apply $\text{Add}(\vec{\sigma}_v, \sigma_{v,i})$ for each $1 \leq i \leq m(v, w_v)$. Thus, the $m(v, w_v)$ routes constructed above for $\vec{\sigma}_v$ have served all the pickup and delivery requests for leaf v, which implies that Condition 2 of Definition 2.2 for a $\langle v, w_v \rangle$-route list is satisfied. Since each route starts and ends at $p(v)$, Condition 1 of Definition 2.2 is also satisfied. Moreover, since $V(T_v) = \{v\}$, Condition 3 of Definition 2.2 is satisfied. Since $\sigma_{v,1}$ has an initial load of w_v, and since the $m(v, w_v)$ routes comply with the greedy structure, Conditions 5 and 6 of Definition 2.2 are also satisfied.

Stage 3 (Finalization): If $m(v, w_v) = n(v)$, then in order to satisfy Conditions 4 and 7 of Definition 2.2, we set $\sigma_{v,n(v)+1} \leftarrow [p(v), 0]$, and apply $\text{Add}(\vec{\sigma}_v, \sigma_{v,n(v)+1})$ to add to $\vec{\sigma}_v$ a degenerate route that consists of only $p(v)$. Otherwise, due to (2.5), we have $m(v, w_v) = n(v) + 1$, which implies that after Stage 2, $\vec{\sigma}_v$ has already satisfied Conditions 4 and 7 of Definition 2.2.

Hence, we have obtained a $\langle v, w_v \rangle$-route list $\vec{\sigma}_v$ for each leaf v. Figure 2.3 illustrates the route lists constructed for each leaf of the instance in Figure 2.2(a). Lemma 2.3 can thus be established.

Lemma 2.3. *For each leaf v of T, a $\langle v, w_v \rangle$-route list $\vec{\sigma}_v$ can be obtained in $O(n(v))$ time.*

Proof. We have shown that $\vec{\sigma}_v$ constructed by the three stages above is a $\langle v, w_v \rangle$-route

list. Moreover, the construction of each $\sigma_{v,i}$ for $1 \leq i \leq m(v, w_v)$ in Stage 2 takes $O(1)$ time. By Lemma 2.2, $\text{Add}(\vec{\sigma}_v, \sigma_{v,i})$ applied in Stages 2 and 3 for each $1 \leq i \leq n(v)+1$ takes $O(1)$ time. Since $m(v, w_v) \leq n(v) + 1$ by (2.5), the construction of $\vec{\sigma}_v$ takes $O(n(v))$ time in total. □

2.5.4 Construction of $\langle v, w_v \rangle$-Route Lists for Internal Nodes

Consider each internal node v of T, with the left and right children denoted by l and r. Since $\mathcal{I}$ is standard, $q(v) = 0$, which implies $q(T_v) = q(T_l) + q(T_r)$.

Given an $\langle l, w_l \rangle$-route list $\vec{\sigma}_l$, which contains $n(l) + 1$ routes denoted by $\sigma_{l,i}$ for $1 \leq i \leq n(l) + 1$, and given an $\langle r, w_r \rangle$-route list $\vec{\sigma}_r$, which contains $n(r) + 1$ routes denoted by $\sigma_{r,i}$ for $1 \leq i \leq n(r) + 1$, we need to construct a $\langle v, w_v \rangle$-route list $\vec{\sigma}_v$ to establish Lemma 2.4.

Lemma 2.4. *For each internal node v of T with the left and right children denoted by l and r, given an $\langle l, w_l \rangle$-route list $\vec{\sigma}_l$ and an $\langle r, w_r \rangle$-route list $\vec{\sigma}_r$, a $\langle v, w_v \rangle$-route list $\vec{\sigma}_v$ can be obtained in $O(n(l) + n(r))$ time.*

Since $\mathcal{I}$ is standard, implying that $q(T_l) \geq q(T_r)$, it is sufficient to consider only the following three cases: Case 1, where $q(T_l) \geq 0$ and $q(T_r) \geq 0$, or $q(T_l) < 0$ and $q(T_r) < 0$; Case 2, where $q(T_l) \geq 0$, $q(T_r) < 0$, and $q(T_v) \geq 0$; and Case 3, where $q(T_l) \geq 0$, $q(T_r) < 0$, and $q(T_v) < 0$. Thus, we present three sub-routines in Section 2.5.4.1, Section 2.5.4.2, and Section 2.5.4.3, to construct $\vec{\sigma}_v$ and prove Lemma 2.4 for each of these three cases, respectively.

2.5.4.1 Case 1: $q(T_l) \geq 0$ and $q(T_r) \geq 0$, or $q(T_l) < 0$ and $q(T_r) < 0$.

As we have shown, $q(T_v) = q(T_l) + q(T_r)$. Thus, in this case, if $q(T_l) \geq 0$ and $q(T_r) \geq 0$, then $q(T_v) \geq 0$; otherwise, $q(T_l) < 0$ and $q(T_r) < 0$, implying that $q(T_v) < 0$.

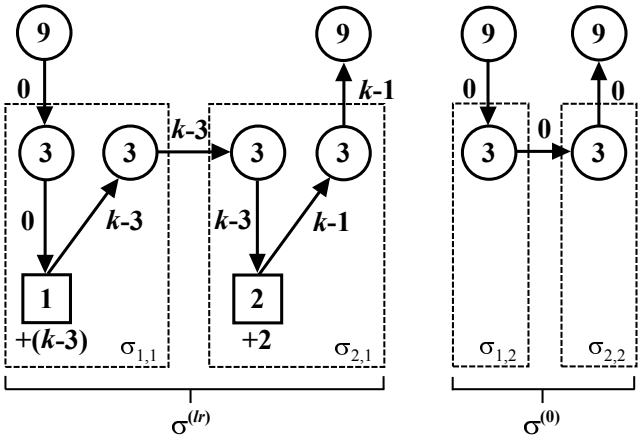


Figure 2.4: Construction of a $\langle 3, w_3 \rangle$-route list for the instance in Figure 2.2(a).

To understand the basic idea of the construction, let us first consider internal node 3 of the instance in Figure 2.2(a), where its left child is leaf 1 and its right child is leaf 2. From Table 2.1, we have $q(T_1) = k - 3 \geq 0$, $q(T_2) = 2 \geq 0$, $w_3 = 0$, $\text{tail}(3, w_3) = k - 1$, and $n(3) = 1$. Given the $\langle 1, w_1 \rangle$-route list $\vec{\sigma}_1$ and $\langle 2, w_2 \rangle$-route list $\vec{\sigma}_2$ shown in Figure 2.3, we can merge $\vec{\sigma}_1$ and $\vec{\sigma}_2$ to construct a $\langle 3, w_3 \rangle$-route list $\vec{\sigma}_3$ as follows. According to Definition 2.2, $\vec{\sigma}_3$ must contain two routes that comply with the greedy structure, with the first route $\sigma_{3,1}$ having an empty initial load and a final load of $k - 1$, and the second route $\sigma_{3,2}$ having empty initial and final loads. It can be seen that the first route $\sigma_{1,1}$ of $\vec{\sigma}_1$ has an empty initial load, and its final load equals the initial load of the first route $\sigma_{2,1}$ of $\vec{\sigma}_2$, which has a final load of $k-1$. Thus, as shown in Figure 2.4, we can obtain a route $\sigma^{(lr)}$ that follows $\sigma_{1,1}$ and $\sigma_{2,1}$ sequentially, so that $\sigma^{(lr)}$ has an empty initial load and a final load of $k - 1$. Thus, $\sigma^{(lr)}$ can be used to form $\sigma_{3,1}$. Furthermore, since both the second route $\sigma_{1,2}$ of $\vec{\sigma}_1$ and the second route $\sigma_{2,2}$ of $\vec{\sigma}_2$ have empty initial and final loads, we can obtain a route $\sigma^{(0)}$ that follows $\sigma_{1,2}$ and $\sigma_{2,2}$ sequentially, so that $\sigma^{(0)}$ has empty initial and final loads. Thus, $\sigma^{(0)}$ can be used to form $\sigma_{3,2}$. Accordingly, by extending both $\sigma_{3,1}$ and $\sigma_{3,2}$ to start and end at vertex 9, we can obtain the two routes of $\vec{\sigma}_3$, which form a $\langle 3, w_3 \rangle$-route list.

By generalizing the idea above, we develop the following sub-routine for the construction of $\vec{\sigma}_v$ for Case 1, consisting of three stages.

Stage 1 (Initialization): Set $\vec{\sigma}_v \leftarrow \emptyset$, and construct two routes, denoted by $\sigma^{(lr)}$ and $\sigma^{(0)}$, in the following two steps:

1. Construction of $\sigma^{(lr)}$: By Definition 2.2 and $w_r = \text{tail}(l, w_l)$, the final load of $\sigma_{l,m(l,w_l)}$ equals the initial load of $\sigma_{r,1}$. Thus, set $\sigma^{(lr)} \leftarrow \text{Link}(\sigma_{l,m(l,w_l)}, \sigma_{r,1})$, so that $\sigma^{(lr)}$ follows $\sigma_{l,m(l,w_l)}$ and $\sigma_{r,1}$ sequentially.

2. Construction of $\sigma^{(0)}$: By Definition 2.2, for each $u \in \{l, r\}$, if $m(u, w_u) = n(u)$, both the initial and final loads of $\sigma_{u,n(u)+1}$ are empty. Thus, set $\sigma^{(0)}$ to be a route that follows $\sigma_{l,n(l)+1}$ if $m(l, w_l) = n(l)$, and then follows $\sigma_{r,n(r)+1}$ if $m(r, w_r) = n(r)$. Thus, $\sigma^{(0)}$ has empty initial and final loads, and if $m(u, w_u) = n(u) + 1$ for each $u \in \{l, r\}$, then $\sigma^{(0)}$ is an empty route.

Accordingly, consider $\sigma_{l,i}$ for $1 \leq i \leq m(l, w_l) - 1$, $\sigma_{r,i}$ for $2 \leq i \leq m(r, w_r)$, $\sigma^{(lr)}$, and $\sigma^{(0)}$. They together serve all the pickup and delivery requests in T_l and T_r. Thus, using these $m(l, w_l) + m(r, w_r)$ routes, the sub-routine constructs $\vec{\sigma}_v$ in the next two stages.

Stage 2 (Construction): The aim is to construct $m(v, w_v)$ routes of $\vec{\sigma}_v$ that comply with the greedy structure, with the first route having an initial load of w_v. We apply the following two steps to construct routes $\sigma_{v,i}$ for $1 \leq i \leq m(l, w_l) + m(r, w_r) - 1$ and add them to $\vec{\sigma}_v$:

1. Set $\sigma_{v,i} \leftarrow \sigma_{l,i}$ for each $1 \leq i \leq m(l, w_l) - 1$. Set $\sigma_{v,m(l,w_l)} \leftarrow \sigma^{(lr)}$. Set $\sigma_{v,m(l,w_l)+i-1} \leftarrow \sigma_{r,i}$ for each $2 \leq i \leq m(r, w_r)$.

2. Apply $\text{Add}(\vec{\sigma}_v, \sigma_{v,i})$ for $1 \leq i \leq m(l, w_l) + m(r, w_r) - 1$.

The construction of the $m(v, w_v)$ routes for $q(T_l) \geq 0$ and $q(T_r) \geq 0$ is illustrated in Figure 2.5, and the construction of routes for $q(T_l) < 0$ and $q(T_r) < 0$ is similar.

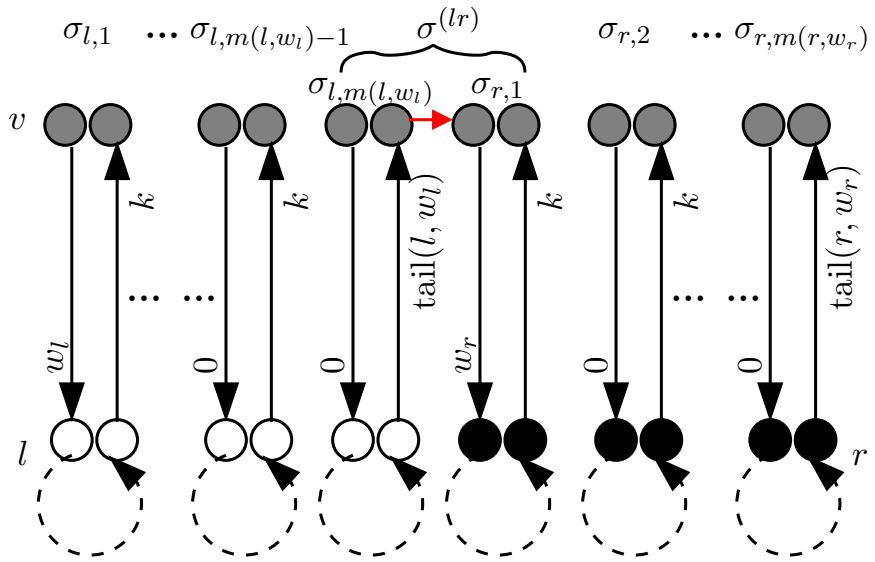


Figure 2.5: Construction of $\vec{\sigma}_v$ for an internal node v: when $q(T_l) \geq 0$ and $q(T_r) \geq 0$.

As a result of Stage 2, since $w_v = w_l$ and $\sigma_{l,1}$ has an initial load of w_l, we obtain that $\sigma_{v,1}$ has an initial load of w_v. Since the first $m(l, w_l)$ routes in $\vec{\sigma}_l$ and the first $m(r, w_r)$ routes in $\vec{\sigma}_r$ together have a total balance of $q(T_l)+q(T_r) = q(T_v)$, the routes $\sigma_{v,i}$ for $1 \leq i \leq m(l, w_l)+m(r, w_r)-1$ also have a total balance of $q(T_v)$. Now consider the following two situations: (i) if $q(T_l) \geq 0$ and $q(T_r) \geq 0$, then $q(T_v) \geq 0$. Thus, since the first $m(l, w_l)$ routes in $\vec{\sigma}_l$ and the first $m(r, w_r)$ routes in $\vec{\sigma}_r$ both comply with the greedy structure, each of the routes $\sigma_{v,i}$ for $1 \leq i \leq m(l, w_l) + m(r, w_r) - 1$ must carry as much of the product as possible out of T_v; (ii) If $q(T_l) < 0$ and $q(T_r) < 0$, then $q(T_v) < 0$. Thus, since the first $m(l, w_l)$ routes in $\vec{\sigma}_l$ and the first $m(r, w_r)$ routes in $\vec{\sigma}_r$ both comply with the greedy structure, each of the routes $\sigma_{v,i}$ for $1 \leq i \leq m(l, w_l) + m(r, w_r) - 1$ must carry as much of the product as possible into T_v. Therefore, in both situations, the $m(l, w_l)+m(r, w_r)-1$ routes comply with the greedy structure. By (2.3) and (2.4), we obtain that $m(l, w_l) + m(r, w_r) - 1 = m(v, w_v)$, and that $\sigma_{v,m(v,w_v)}$ has a final load of tail(v, w_v). Hence, $\vec{\sigma}_v$ satisfies Conditions 5 and 6 of Definition 2.2 for a $\langle v, w_v\rangle$-route list.

Moreover, due to Condition 1 of Definition 2.2 for $\vec{\sigma}_l$ and $\vec{\sigma}_r$, each of the routes

$\sigma_{v,i}$ constructed in Stage 2 for $1 \leq i \leq m(v, w_v)$, as well as the route $\sigma^{(0)}$ constructed in Stage 1, must start and end at v, visiting vertices in T_v only. Due to Conditions 2 and 3 of Definition 2.2 for $\vec{\sigma}_l$ and $\vec{\sigma}_r$, the route $\sigma^{(0)}$ and the $m(v, w_v)$ routes in $\vec{\sigma}_v$ together serve all the pickup and delivery requests in T_v and traverse each edge $e(u)$ with $u \in V(T_v) \setminus \{v\}$ at most $2n(u) + 2$ times, satisfying Conditions 2 and 3 of Definition 2.2.

Stage 3 (Finalization): The aim is to transform $\vec{\sigma}_v$ to satisfy Conditions 1, 2, 3, 4 and 7 of Definition 2.2 to become a $\langle v, w_v \rangle$-route list. If $m(v, w_v) = n(v)$, then we add $\sigma^{(0)}$ to $\vec{\sigma}_v$, so that $|\vec{\sigma}_v| = n(v) + 1$. Since $\sigma^{(0)}$ has empty initial and final loads, Conditions 2, 3, 4 and 7 of Definition 2.2 are thus satisfied. Otherwise, $m(v, w_v) = n(v) + 1$, and we need to link $\sigma^{(0)}$ with a route in $\vec{\sigma}_v$ that has an empty initial load or empty final load. To see that such a route always exists in $\vec{\sigma}_v$, note that $|\vec{\sigma}_v| = m(v, w_v) \geq 2$, which implies that $\vec{\sigma}_v$ contains at least two routes. Thus, since routes in $\vec{\sigma}_v$ comply with the greedy structure, $\sigma_{v,2}$ must have an empty initial load if $q(T_v) \geq 0$, or else $\sigma_{v,1}$ must have an empty final load. As a result, Conditions 2, 3, 4 and 7 of Definition 2.2 are satisfied. To further satisfy Condition 1 of Definition 2.2, we can apply Extend$(\vec{\sigma}_v, v)$ to extend each route in $\vec{\sigma}_v$ to start and end at $p(v)$. Thus, the transformation of $\vec{\sigma}_v$ can be summarized into the following two steps:

1. If $m(v, w_v) = n(v)$, then Add$(\vec{\sigma}_v, \sigma^{(0)})$. If $m(v, w_v) \neq n(v)$ and $q(T_v) \geq 0$, then apply Replace$(\vec{\sigma}_v, 2,$ Link$(\sigma^{(0)}, \sigma_{v,2}))$ to replace $\sigma_{v,2}$ with the route that links $\sigma^{(0)}$ and $\sigma_{v,2}$. Otherwise, we have $m(v, w_v) \neq n(v)$ and $q(T_v) < 0$, and thus, apply Replace$(\vec{\sigma}_v, 1,$ Link$(\sigma_{v,1}, \sigma^{(0)}))$ to replace $\sigma_{v,1}$ with the route that links $\sigma_{v,1}$ and $\sigma^{(0)}$.

2. Apply Extend$(\vec{\sigma}_v, v)$ to extend each route in $\vec{\sigma}_v$ to start and end at $p(v)$.

Hence, according to Definition 2.2, $\vec{\sigma}_v$ returned by the above 3-stage sub-routine is a $\langle v, w_v \rangle$-route list. Moreover, by Lemma 2.2, the Link operator takes $O(1)$ time,

which implies that Stage 1 runs in $O(1)$ time. Since the Add operator takes $O(1)$ time, and since $|\vec{\sigma}_l| = n(l) + 1$ and $|\vec{\sigma}_r| = n(r) + 1$, Stage 2 runs in $O(n(l) + n(r))$ time. Since the Add and Link operators take $O(1)$ time, since the Replace and Extend operators on $\vec{\sigma}_v$ take $O(|\vec{\sigma}_v|)$ time, and since $|\vec{\sigma}_v| = n(v) + 1$, Stage 3 runs in $O(n(v))$ time. By (2.1) and $q(T_v) = q(T_r) + q(T_l)$, we have $n(v) \leq n(l) + n(r)$. Thus, all three stages run in $O(n(l) + n(r))$ time. Lemma 2.4 has thus been proved for Case 1.

2.5.4.2 Case 2: $q(T_l) \geq 0$, $q(T_r) < 0$, and $q(T_v) \geq 0$.

In this case, to achieve a total balance of $q(T_v)$, routes to be constructed for $\vec{\sigma}_v$ need to carry $q(T_v)$ more units of the product out of T_v than into T_v.

To understand the basic idea of the construction, let us first consider internal node 6 of the instance in Figure 2.2(a), where its left child is leaf 4 and its right child is leaf 5. From Table 2.1, we have $q(T_4) = 2 \geq 0$, $q(T_5) = -1 < 0$, $w_6 = k - 1$, $\text{tail}(6, w_6) = k$, and $n(6) = 1$. Given the $\langle 4, w_4 \rangle$-route list $\vec{\sigma}_4$ and $\langle 5, w_5 \rangle$-route list $\vec{\sigma}_5$ shown in Figure 2.3, we can merge $\vec{\sigma}_4$ and $\vec{\sigma}_5$ to construct a $\langle 6, w_6 \rangle$-route list $\vec{\sigma}_6$ as follows. According to Definition 2.2, $\vec{\sigma}_6$ must contain two routes that comply with the greedy structure, with the first route $\sigma_{6,1}$ having an initial load of $k-1$ and a final load of k, and the second route $\sigma_{6,2}$ having empty initial and final loads. It can be seen that the final load of $\sigma_{4,2}$ equals the initial load of $\sigma_{5,1}$, and thus we can obtain a route $\sigma^{(lr)}$ that follows $\sigma_{4,2}$ and $\sigma_{5,1}$ sequentially, as shown in Figure 2.6. Having noted that both $\sigma^{(lr)}$ and $\sigma_{5,2}$ have empty initial and final loads, we can construct $\sigma^{(0)}$ by following $\sigma_{5,2}$ and $\sigma^{(lr)}$ sequentially, so that $\sigma^{(0)}$ has empty initial and final loads. Thus, $\sigma^{(0)}$ can be used to form $\sigma_{6,2}$. Moreover, since $\sigma_{4,1}$ has an initial load of $k - 1$ and a final load of k, it can be used to form $\sigma_{6,1}$. Thus, by extending both $\sigma_{6,1}$ and $\sigma_{6,2}$ to start and end at vertex 8, we can obtain the two routes of $\vec{\sigma}_6$, which form a $\langle 6, w_6 \rangle$-route list.

By generalizing the idea above, we develop the following sub-routine for the con-

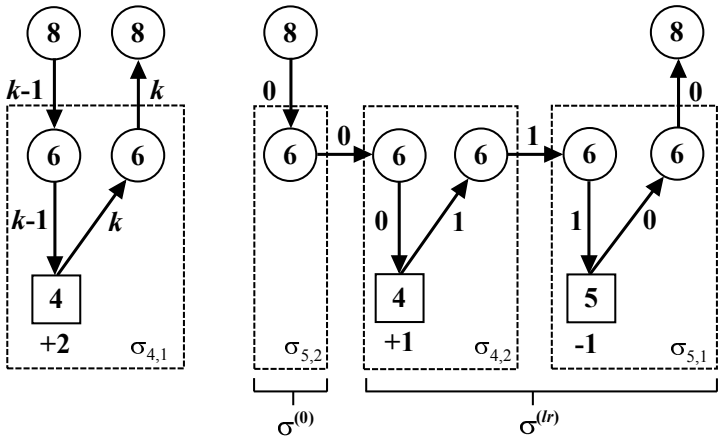


Figure 2.6: Construction of a $\langle 6, w_6 \rangle$-route list for the instance in Figure 2.2(a).

struction of $\vec{\sigma}_v$ for Case 2, which, as with Case 1, consists of three stages.

Stage 1 (Initialization): Repeat Stage 1 in Case 1 to set $\vec{\sigma}_v \leftarrow \emptyset$, and construct routes $\sigma^{(lr)}$ and $\sigma^{(0)}$.

Stage 2 (Construction): This has the same objective as Stage 2 in Case 1, that of constructing $m(v, w_v)$ routes of $\vec{\sigma}_v$ that comply with the greedy structure, with the first route having an initial load of w_v. However, the construction is more complicated, and $\sigma^{(0)}$ may need modifications so that $\sigma^{(0)}$ and the $m(v, w_v)$ routes of $\vec{\sigma}_v$ together serve all the pickup and delivery requests in T_v and traverse each edge $e(u)$ with $u \in V(T_v) \setminus \{v\}$ at most $2n(u) + 2$ times, with $\sigma^{(0)}$ still having empty initial and final loads.

First, consider a simple situation, where $m(r, w_r) = 1$. If $m(l, w_l) = 1$, then $\sigma^{(lr)}$ alone achieves a balance of $q(T_l) + q(T_r) = q(T_v)$, complies with the greedy structure, and has an initial load of w_v. Thus, we apply $\text{Add}(\vec{\sigma}_v, \sigma^{(lr)})$, so that $\sigma_{v,1} \leftarrow \sigma^{(lr)}$. Otherwise, $m(l, w_l) \geq 2$, which implies that $\sigma^{(lr)}$ has an empty initial load. Thus, consider the following two cases: (i) If the final load of $\sigma^{(lr)}$ is not empty, then consider routes $\sigma_{l,i}$ for $1 \leq i \leq m(l, w_l) - 1$, and route $\sigma^{(lr)}$, which together comply with the greedy structure, with $\sigma_{l,1}$ having an initial load of $w_l = w_v$. Thus, we apply $\text{Add}(\vec{\sigma}_v, \sigma_{l,i})$, so that $\sigma_{v,i} \leftarrow \sigma_{l,i}$, for $1 \leq i \leq m(l, w_l) - 1$, and then apply $\text{Add}(\vec{\sigma}_v, \sigma^{(lr)})$,

so that $\sigma_{v,m(l,w_l)} \leftarrow \sigma^{(lr)}$. (ii) If $\sigma^{(lr)}$ has an empty final load, then $\text{tail}(r, w_r) = 0$, which, together with $w_r = \text{tail}(l, w_l)$, $m(r, w_r) = 1$, $q(T_r) < 0$, (2.3), and (2.4), implies that $\text{tail}(l, w_l) + q(T_r) = 0$. Accordingly, routes $\sigma_{l,i}$ for $1 \leq i \leq m(l, w_l) - 1$ have a total balance of $q(T_l) - \text{tail}(l, w_l) = q(T_v) - q(T_r) - \text{tail}(l, w_l) = q(T_v)$, and thus comply with the greedy structure, with $\sigma_{l,1}$ having an initial load of $w_l = w_v$. We then apply $\text{Add}(\vec{\sigma}_v, \sigma_{l,i})$, so that $\sigma_{v,i} \leftarrow \sigma_{l,i}$, for $1 \leq i \leq m(l, w_l) - 1$. Moreover, since $\sigma^{(lr)}$ has empty initial and final loads, we reset $\sigma^{(0)} \leftarrow \text{Link}(\sigma^{(0)}, \sigma^{(lr)})$, which still retains the empty initial and final loads.

Next, consider another situation, where $m(r, w_r) \geq 2$. The construction of the $m(v, w_v)$ routes has the following three steps, which are illustrated in Figure 2.7.

In Step 1, we reset $\sigma^{(0)}$ as follows by linking $\sigma^{(0)}$ and $\sigma^{(lr)}$:

- Set $\sigma^{(0)} \leftarrow \text{Link}(\sigma^{(0)}, \sigma^{(lr)})$.

To prove that this Link operation is valid and that $\sigma^{(0)}$ still has empty initial and final loads, it is sufficient to show that $\sigma^{(lr)}$ has empty initial and final loads. Define $\Delta_m := m(l, w_l) - m(r, w_r)$. By $q(T_v) \geq 0$ and Lemma 2.5 below, $\Delta_m \geq 0$, implying $m(l, w_l) \geq m(r, w_r) \geq 2$. Since the first $m(l, w_l)$ routes in $\vec{\sigma}_l$ and the first $m(r, w_r)$ routes in $\vec{\sigma}_r$ both comply with the greedy structure, $\sigma_{l,m(l,w_l)}$ must have an empty initial load, and $\sigma_{r,1}$ must have an empty final load. Thus, $\sigma^{(lr)}$ has empty initial and final loads.

Lemma 2.5. *For each internal node v of T, if $q(T_v) \geq 0$, then $\Delta_m \geq 0$, and otherwise $\Delta_m \leq 0$.*

Proof. By $w_r = \text{tail}(l, w_l) \leq k$ and $q(T_r) < 0$, we have $q(T_r) + w_r < k$. Thus, we have

$$\text{tail}(r, w_r) = [q(T_r) + w_r] \pmod{k} \leq k - 1, \tag{2.6}$$

$$-q(T_r) = [m(r, w_r) - 1]k + w_r - \text{tail}(r, w_r), \tag{2.7}$$

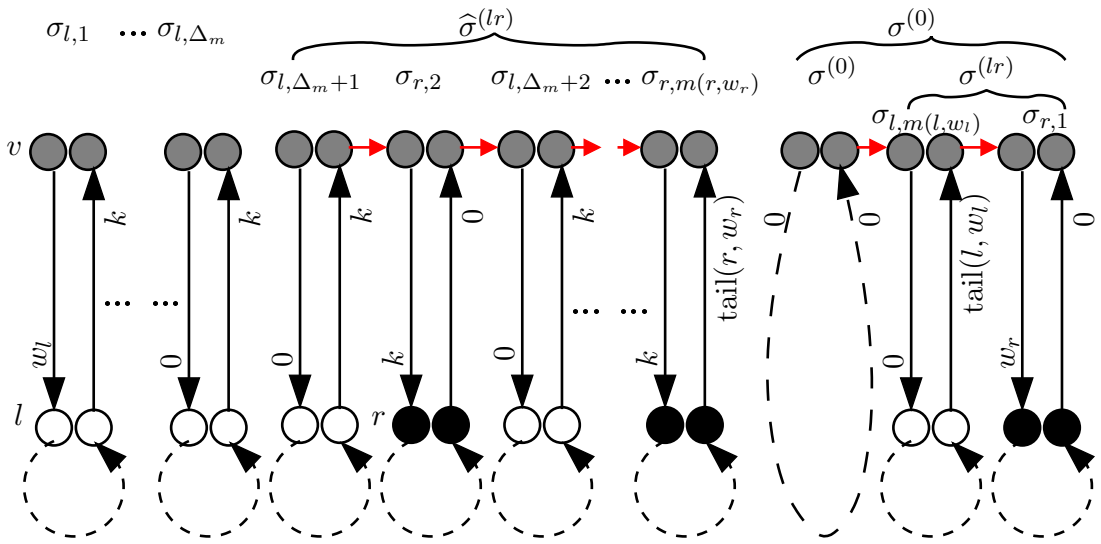


Figure 2.7: Construction of $\vec{\sigma}_v$ for an internal node v: when $q(T_l) \geq 0$, $q(T_r) < 0$, $q(T_v) \geq 0$, and $m(r, w_r) \geq 2$.

due to Lemma 2.1. By $q(T_l) \geq 0$, (2.3), and (2.4), we have:

$$q(T_l) \quad = \quad [m(l, w_l) - 1]k + \text{tail}(l, w_l) - w_l. \tag{2.8}$$

Thus, since $q(T_v) = q(T_l) + q(T_r)$, from $w_r = \text{tail}(l, w_l)$, $w_l = w_v$, (2.7), and (2.8), we obtain:

$$q(T_v) + w_v = q(T_l) + q(T_r) + w_l = [m(l, w_l) - m(r, w_r)]k + \text{tail}(r, w_r). \tag{2.9}$$

Moreover, if $q(T_v) \geq 0$, then $\text{tail}(r, w_r) \leq k - 1$ by (2.6), which, together with $w_v \geq 0$ and (2.9), implies that $m(l, w_l) - m(r, w_r) \geq 0$, and thus $\Delta_m \geq 0$. Otherwise, $q(T_v) < 0$, which, together with $w_v \leq k$, $\text{tail}(r, w_r) \geq 0$, and (2.9), implies that $m(l, w_l) - m(r, w_r) \leq 0$, and thus $\Delta_m \leq 0$. □

In Step 2, as shown below and in Figure 2.7, noting that $m(l, w_l) \geq m(r, w_r)$, we extract $m(r, w_r) - 1$ routes from $\vec{\sigma}_l$ and from $\vec{\sigma}_r$, respectively, and then link them to construct a route, denoted by $\widehat{\sigma}^{(lr)}$, which follows σ_{l,Δ_m+1}, $\sigma_{r,2}$,σ_{l,Δ_m+2}, $\sigma_{r,3}$, ...,

$\sigma_{l,\Delta_m+m(r,w_r)-1}$, and $\sigma_{r,m(r,w_r)}$, sequentially, moving amounts of the product from T_l to T_r back and forth:

- Set $\vec{\sigma}'_l \leftarrow \text{Extract}(\vec{\sigma}_l, \Delta_m+1, \Delta_m+m(r,w_r)-1)$, and $\vec{\sigma}'_r \leftarrow \text{Extract}(\vec{\sigma}_r, 2, m(r,w_r))$. Set $\widehat{\sigma}^{(lr)} \leftarrow \text{CrossLink}\ (\vec{\sigma}'_l, \vec{\sigma}'_r)$.

To show that the CrossLink operation is valid, consider the first $m(l,w_l)$ routes in $\vec{\sigma}_l$ and the first $m(r,w_r)$ routes in $\vec{\sigma}_r$, which both comply with the greedy structure. Thus, for $1 \le i \le m(r,w_r)-1$, both the final load of σ_{l,Δ_m+i} and the initial load of $\sigma_{r,i+1}$ equal k, and for $1 \le i \le m(r,w_r)-2$, both the final load of $\sigma_{r,i+1}$ and the initial load of σ_{l,Δ_m+i+1} are empty. This implies the validity of the CrossLink operation.

In Step 3, using $\widehat{\sigma}^{(lr)}$ and $\sigma_{l,i}$ for $1 \le i \le \Delta_m$, we determine the routes for $\vec{\sigma}_v$ as follows. Since only the first Δ_m routes of $\vec{\sigma}_l$ have not been included in either $\widehat{\sigma}^{(lr)}$ or $\sigma^{(0)}$, we first apply $\text{Add}(\vec{\sigma}_v, \sigma_{l,i})$ for $1 \le i \le \Delta_m$ to add them to $\vec{\sigma}$, so that $\sigma_{v,i} \leftarrow \sigma_{l,i}$ for $1 \le i \le \Delta_m$. However, these routes may not be sufficient to achieve a total balance of $q(T_v)$. Thus, we next determine whether or not to add $\widehat{\sigma}^{(lr)}$, by considering the following two situations: (i) If $\Delta_m = 0$, or the final load of $\widehat{\sigma}^{(lr)}$ is not empty, then apply $\text{Add}(\vec{\sigma}_v, \widehat{\sigma}^{(lr)})$, so that $\sigma_{v,\Delta_m+1} \leftarrow \widehat{\sigma}^{(lr)}$, from which it can be seen that the (Δ_m+1) routes in $\vec{\sigma}_v$ together have a total balance of $q(T_l)+q(T_r) = q(T_v)$ and comply with the greedy structure, and that route $\sigma_{v,1}$ has an initial load of $w_l = w_v$. (ii) If $\Delta_m > 0$ and $\widehat{\sigma}^{(lr)}$ has an empty final load, then since $\Delta_m > 0$, the initial load of $\widehat{\sigma}^{(lr)}$ is also empty. Thus, reset $\sigma^{(0)} \leftarrow \text{Link}(\sigma^{(0)}, \widehat{\sigma}^{(lr)})$, so that $\sigma^{(0)}$ still has empty initial and final loads. It can be seen that the Δ_m routes in $\vec{\sigma}_v$ together have a total balance of $q(T_l)+q(T_r) = q(T_v)$ and comply with the greedy structure, and that route $\sigma_{v,1}$ has an initial load of $w_l = w_v$. Accordingly, Step 3 can be summarized as follows:

- Apply $\text{Add}(\vec{\sigma}_v, \sigma_{l,i})$ for $1 \le i \le \Delta_m$. If $\Delta_m = 0$ or the final load of $\widehat{\sigma}^{(lr)}$ is not empty, then apply $\text{Add}(\vec{\sigma}_v, \widehat{\sigma}^{(lr)})$; otherwise, reset $\sigma^{(0)} \leftarrow \text{Link}(\sigma^{(0)}, \widehat{\sigma}^{(lr)})$.

As a result of Stage 2, routes in $\vec{\sigma}_v$ achieve a total balance of $q(T_v)$, and comply

with the greedy structure, with the first route having an initial load of w_v. By (2.3) and (2.4), we obtain that $|\vec{\sigma}_v| = m(v, w_v)$ and $\sigma_{v,m(v,w_v)}$ has a final load of tail(v, w_v). Hence, $\vec{\sigma}_v$ satisfies Conditions 5 and 6 of Definition 2.2 for a $\langle v, w_v \rangle$-route list. Moreover, as with Stage 2 for Case 1, it can be seen that $\sigma^{(0)}$ still has empty initial and final loads, and that all routes in $\vec{\sigma}_v \cup \{\sigma^{(0)}\}$ start and end at v, visit vertices in T_v only, and together satisfy Conditions 2 and 3 of Definition 2.2.

Stage 3 (Finalization): Similar to Stage 3 in Case 1, $\vec{\sigma}_v$ can be transformed to satisfy Conditions 1, 2, 3, 4 and 7 of Definition 2.2 by the following two steps:

1. If $m(v, w_v) = n(v)$, then Add$(\vec{\sigma}_v, \sigma^{(0)})$. Otherwise, $\vec{\sigma}_v$ must contain at least two routes with $\sigma_{v,2}$ having an empty initial load, and thus, we can apply Replace$(\vec{\sigma}_v, 2, \text{Link}(\sigma^{(0)}, \sigma_{v,2}))$ to replace $\sigma_{v,2}$ with the route that links $\sigma^{(0)}$ and $\sigma_{v,2}$. As a result, $\vec{\sigma}_v$ satisfies Conditions 2, 3, 4 and 7 of Definition 2.2.

2. Apply Extend$(\vec{\sigma}_v, v)$ to extend each route in $\vec{\sigma}_v$ to start and end at $p(v)$. As a result, $\vec{\sigma}_v$ satisfies Condition 1 of Definition 2.2.

Hence, according to Definition 2.2, $\vec{\sigma}_v$, returned by the above 3-stage sub-routine, is a $\langle v, w_v \rangle$-route list. Moreover, as with Case 1, it can be shown that Stage 1 runs in $O(1)$ time and Stage 3 runs in $O(n(v))$ time. Now consider Stage 2: If $m(r, w_r) = 1$, then since the Add and Link operators run in $O(1)$ time (due to Lemma 2.2), and since $m(l, w_l) \leq n(l) + 1$, Stage 2 runs in $O(n(l))$ time. Otherwise, $m(r, w_r) \geq 2$, and Stage 2 has three steps. It is easy to see that Step 1 and Step 3 run in $O(1)$ time and $O(n(l) + n(r))$ time, respectively. For Step 2, we have $|\vec{\sigma}'_l| + |\vec{\sigma}'_r| \leq |\vec{\sigma}_l| + |\vec{\sigma}_r| \leq n(l) + n(r) + 2$. Thus, since the Extract operator on $\vec{\sigma}_l$ and on $\vec{\sigma}_r$ run in $O(|\vec{\sigma}_l|)$ time and $O(|\vec{\sigma}_r|)$ time, respectively, and since the CrossLink operator on $\vec{\sigma}'_l$ and $\vec{\sigma}'_r$ runs in $O(|\vec{\sigma}'_l| + |\vec{\sigma}'_r|)$ time, then Step 2 runs in $O(n(l) + n(r))$ time, and so does Stage 2. By (2.1) and $q(T_v) = q(T_r) + q(T_l)$, we have $n(v) \leq n(l) + n(r)$. Thus, the three stages together run in $O(n(l) + n(r))$ time. Lemma 2.4 has been proved for Case 2.

2.5.4.3 Case 3: $q(T_l) \geq 0$, $q(T_r) < 0$, and $q(T_v) < 0$.

In this case, to achieve a total balance of $q(T_v)$, routes to be constructed for $\vec{\sigma}_v$ need to carry $-q(T_v)$ more units of the product into T_v than out of T_v.

To understand the basic idea of the construction, let us first consider internal node 8 of the instance in Figure 2.2(a), where its left child is leaf 6 and its right child is internal node 7. From Table 2.1, we have $q(T_6) = 1 \geq 0$, $q(T_7) = -k < 0$, $q(T_8) = 1 - k < 0$, $w_8 = k - 1$, $\text{tail}(8, w_8) = 0$, and $n(8) = 1$. Given the $\langle 6, w_6 \rangle$-route list $\vec{\sigma}_6$ shown in Figure 2.6 and the $\langle 7, w_7 \rangle$-route list $\vec{\sigma}_7$ shown in Figure 2.3, we can merge $\vec{\sigma}_6$ and $\vec{\sigma}_7$ to construct a $\langle 8, w_8 \rangle$-route list $\vec{\sigma}_8$ as follows. According to Definition 2.2, $\vec{\sigma}_8$ must contain two routes that comply with the greedy structure, with the first route $\sigma_{8,1}$ having an initial load of $k - 1$ and an empty final load, and the second route $\sigma_{8,2}$ having empty initial and final loads. It can be seen that the final load of $\sigma_{6,1}$ equals the initial load of $\sigma_{7,1}$, and thus we can obtain a route $\sigma^{(lr)}$ that follows $\sigma_{6,1}$ and $\sigma_{7,1}$ sequentially, as shown in Figure 2.8, which has an initial load of $k - 1$ and an empty final load. Thus, $\sigma^{(lr)}$ can be used to form $\sigma_{8,1}$. Furthermore, since both $\sigma_{6,2}$ and $\sigma_{7,2}$ have empty initial and final loads, we can then construct a route $\sigma^{(0)}$ by following $\sigma_{6,2}$ and $\sigma_{7,2}$ sequentially, so that $\sigma^{(0)}$ has empty initial and final loads. Thus, $\sigma^{(0)}$ can be used to form $\sigma_{8,2}$. Accordingly, by extending both $\sigma_{8,1}$ and $\sigma_{8,2}$ to start and end at vertex 9, we can obtain the two routes of $\vec{\sigma}_8$, and can verify these forming a $\langle 8, w_8 \rangle$-route list.

By generalizing the above idea, we develop the following sub-routine for the construction of $\vec{\sigma}_v$ for Case 3, which, as with Case 1 and Case 2, consists of three stages.

Stage 1 (Initialization): Repeat Stage 1 in Case 1 to set $\vec{\sigma}_v \leftarrow \emptyset$, and construct routes $\sigma^{(lr)}$ and $\sigma^{(0)}$.

Stage 2 (Construction): Similar to Stage 2 in Case 2, the aim is to construct $m(v, w_v)$ routes of $\vec{\sigma}_v$ that comply with the greedy structure, with the first route in $\vec{\sigma}_v$ having an initial load of w_v, and with route $\sigma^{(0)}$ retaining its empty initial and

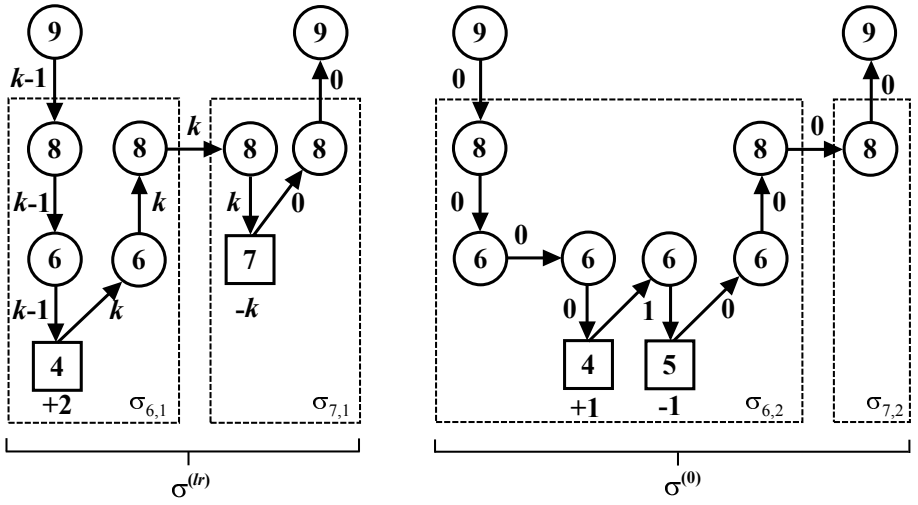


Figure 2.8: Construction of a $\langle 8, w_8 \rangle$-route list for the instance in Figure 2.2(a).

final loads. .

First, consider a simple situation, where $m(l, w_l) = 1$. Since route $\sigma^{(lr)}$ and routes $\sigma_{r,i}$ for $2 \leq i \leq m(r, w_r) - 1$ together comply with the greedy structure, with $\sigma^{(lr)}$ having an initial load of $w_l = w_v$, we apply $\text{Add}(\vec{\sigma}_v, \sigma^{(lr)})$, so that $\sigma_{v,1} \leftarrow \sigma^{(lr)}$, and apply $\text{Add}(\vec{\sigma}_v, \sigma_{r,i})$, so that $\sigma_{v,i} \leftarrow \sigma_{r,i}$, for $2 \leq i \leq m(r, w_r)$.

Next, consider another situation, where $m(l, w_l) \geq 2$. Similar to the situation of $m(r, w_l) \geq 2$ in Case 2, the construction of the $m(v, w_v)$ routes has three steps, as illustrated in Figure 2.9.

In Step 1, we reset $\sigma^{(0)} \leftarrow \text{Link}(\sigma^{(0)}, \sigma^{(lr)})$. To prove that Step 1 is valid and that $\sigma^{(0)}$ retains its empty initial and final loads, it is sufficient to show that $\sigma^{(lr)}$ has empty initial and final loads. Recall that $\Delta_m := m(l, w_l) - m(r, w_r)$. By $q(T_v) < 0$ and Lemma 2.5, we have $\Delta_m \leq 0$. Thus, $m(r, w_r) \geq m(l, w_l) \geq 2$. Note that the first $m(l, w_l)$ routes in $\vec{\sigma}_l$, and the first $m(r, w_r)$ routes in $\vec{\sigma}_r$, both comply with the greedy structure. Thus, $\sigma_{l,m(l,w_l)}$ must have an empty initial load, and $\sigma_{r,1}$ must have an empty final load. Hence, $\sigma^{(lr)}$ has empty initial and final loads.

In Step 2, since $m(r, w_r) \geq m(l, w_l)$, we extract $m(l, w_l) - 1$ routes from $\vec{\sigma}_l$ and from $\vec{\sigma}_r$, respectively, and link them so as to construct a route $\widehat{\sigma}^{(lr)}$, which follows

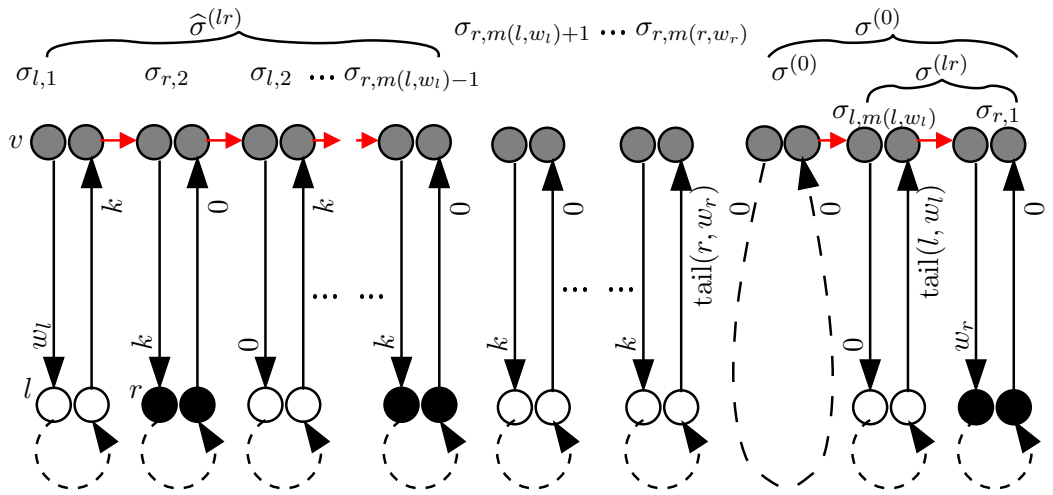


Figure 2.9: Construction of $\vec{\sigma}_v$ for an internal node v: when $q(T_l) \geq 0$, $q(T_r) < 0$, $q(T_v) < 0$, and $m(l, w_l) \geq 2$.

$\sigma_{l,1}, \sigma_{r,2}, \sigma_{l,2}, \sigma_{r,3}$, ..., $\sigma_{l,m(l,w_l)-1}$, and $\sigma_{r,m(l,w_l)}$, sequentially. This can be achieved by setting $\vec{\sigma}'_l \leftarrow \text{Extract}(\vec{\sigma}_l, 1, m(l, w_l) - 1)$, $\vec{\sigma}'_r \leftarrow \text{Extract}(\vec{\sigma}_r, 2, m(l, w_l))$, and $\widehat{\sigma}^{(lr)} \leftarrow$ CrossLink $(\vec{\sigma}'_l, \vec{\sigma}'_r)$. The CrossLink operation is valid, since the first $m(l, w_l)$ routes in $\vec{\sigma}_l$, and the first $m(r, w_r)$ routes in $\vec{\sigma}_r$, both comply with the greedy structure, which implies that for $1 \leq i \leq m(l, w_l) - 1$, both the final load of $\sigma_{l,i}$ and the initial load of $\sigma_{r,i+1}$ are equal to k, and that for $1 \leq i \leq m(l, w_l) - 2$, both the final load of $\sigma_{r,i+1}$ and the initial load of $\sigma_{l,i+1}$ are empty.

In Step 3, since $\widehat{\sigma}^{(lr)}$ starts with $\sigma_{l,1}$, it has an initial load of $w_l = w_v$. Moreover, route $\widehat{\sigma}^{(lr)}$ and the $-\Delta_m$ routes $\sigma_{r,m(l,w_l)+i}$ for $1 \leq i \leq -\Delta_m$ of $\vec{\sigma}_r$ together have a total balance of $q(T_l)+q(T_r) = q(T_v)$ and comply with the greedy structure. Thus, we apply Add$(\vec{\sigma}_v, \widehat{\sigma}^{(lr)})$ to set $\sigma_{v,1} \leftarrow \widehat{\sigma}^{(lr)}$, and apply Add$(\vec{\sigma}_v, \sigma_{r,m(l,w_l)+i})$ to set $\sigma_{v,i+1} \leftarrow \sigma_{r,m(l,w_l)+i}$, for $1 \leq i \leq -\Delta_m$.

As a result of Stage 2, routes in $\vec{\sigma}_v$ have a total balance of $q(T_v)$, and comply with the greedy structure, with the first route having an initial load of w_v. By (2.3) and (2.4), we obtain that $|\vec{\sigma}_v| = m(v, w_v)$ and that $\sigma_{v,m(v,w_v)}$ has a final load of tail(v, w_v). Hence, $\vec{\sigma}_v$ satisfies Conditions 5 and 6 of Definition 2.2 for a $\langle v, w_v \rangle$-route list. Moreover, as with Stage 2 for Cases 1 and 2, it can be seen that $\sigma^{(0)}$ still has

empty initial and final loads, and that all routes in $\vec{\sigma}_v \cup \{\sigma^{(0)}\}$ start and end at v, visit vertices in T_v only, and together satisfy Conditions 2 and 3 of Definition 2.2.

Stage 3 (Finalization): Similar to Stage 3 in Cases 1 and 2, $\vec{\sigma}_v$ can be transformed to satisfy Conditions 1, 2, 3, 4 and 7 of Definition 2.2 by the following two steps:

1. If $m(v, w_v) = n(v)$, then $\text{Add}(\vec{\sigma}_v, \sigma^{(0)})$. Otherwise, $\vec{\sigma}_v$ must contain at least two routes with $\sigma_{v,1}$ having an empty final load, and thus, we can apply $\text{Replace}(\vec{\sigma}_v, 1, \text{Link}(\sigma_{v,1}, \sigma^{(0)}))$ to replace $\sigma_{v,1}$ with the route that links $\sigma_{v,1}$ and $\sigma^{(0)}$. As a result, $\vec{\sigma}_v$ satisfies Conditions 2, 3, 4 and 7 of Definition 2.2.

2. Apply $\text{Extend}(\vec{\sigma}_v, v)$ to extend each route in $\vec{\sigma}_v$ to start and end at $p(v)$. As a result, $\vec{\sigma}_v$ satisfies Condition 1 of Definition 2.2.

Hence, according to Definition 2.2, $\vec{\sigma}_v$ returned by the above 3-stage sub-routine is a $\langle v, w_v \rangle$-route list. Moreover, as with Case 1, it can be shown that Stage 1 runs in $O(1)$ time, Stage 3 runs in $O(n(v))$ time, Stage 2 runs in $O(n(l) + n(r))$ time, and $n(v) \leq n(l) + n(r)$. Thus, the three stages together run in $O(n(l) + n(r))$ time. Lemma 2.4 has been proved for Case 3, which completes the proof of Lemma 2.4.

2.5.5 Main Algorithm

Given the constructions in Section 2.5.3 and Section 2.5.4, we can now develop an approximation algorithm for the CTSPPD-T, as shown in Algorithm 2.1.

Algorithm 2.1.

1. Re-label vertices of V by $1, 2, ..., |V|$, according to the postorder traversal sequence of T.

2. Follow Section 2.5.1 to compute w_v for each $v \in V$ so that $w_s = 0$, and that $w_l = w_v$ and $w_r = \text{tail}(l, w_l)$ for each internal vertex $v \in V$ and its left and right children l and r.

3. For $v = 1, 2, ..., |V|$, do the following:

   (a) If v is a leaf, follow the sub-routine in Section 2.5.3 to construct a $\langle v, w_v \rangle$-route list $\vec{\sigma}_v$.

   (b) Otherwise, v is an internal node. Let $l < v$ and $r < v$ denote the left and right children of v. Given $\vec{\sigma}_l$ and $\vec{\sigma}_r$, which were constructed for l and r before the construction of $\vec{\sigma}_v$, follow the sub-routines in Section 2.5.4 to construct a $\langle v, w_v \rangle$-route list $\vec{\sigma}_v$.

4. Follow the proof of Theorem 2.2 to transform $\vec{\sigma}_s$ to a feasible route σ, and return σ. □

For example, consider the instance shown in Figure 2.2(a). Following Algorithm 2.1, we can construct $\langle v, w_v \rangle$-route lists for $v = 1, 2, ..., 9$. The route lists for $1 \leq v \leq 8$ have been presented in Figures 2.3, 2.4, 2.6, and 2.8. Given the $\langle 3, w_3 \rangle$-route list in Figure 2.4 and the $\langle 8, w_8 \rangle$-route list in Figure 2.8, we can construct a $\langle 9, w_9 \rangle$-route list $\vec{\sigma}_9$ by following the sub-routine in Section 2.5.4.2, since $q(T_3) = k - 1 > 0$, $q(T_8) = 1 - k < 0$, and $q(T_9) = 0$. Accordingly, the two routes in $\vec{\sigma}_9$ can be linked together to form a feasible route, this being the same as the route in Figure 2.2(b).

The approximation ratio and time complexity of Algorithm 2.1 are shown in Theorem 2.3.

Theorem 2.3. *Given any standard instance $\mathcal{I}$, Algorithm 2.1 returns a feasible route σ with $d(\sigma) \leq 2\text{OPT}(\mathcal{I})$ in $O(|V|(1 + \sum_{v \in V} |q(v)|/k))$ time, and the approximation ratio of 2 is tight.*

Proof. To show the correctness and approximation ratio of Algorithm 2.1, we first prove by induction that $\vec{\sigma}_v$ obtained in Step 3 of Algorithm 2.1 for each $v = 1, 2, ..., |V|$ is a $\langle v, w_v \rangle$-route list. This holds true when v is a leaf of T, due to Step 3(a) and Lemma 2.3. For each internal node v with the left and right children denoted by $l < v$

and $r < v$, suppose that $\vec{\sigma}_t$ is a $\langle t, w_t \rangle$-route list for $1 \leq t \leq v-1$, which includes $t = l$ and $t = r$. Thus, due to Step 3(b) and Lemma 2.4, $\vec{\sigma}_v$ is a $\langle v, w_v \rangle$-route list. The induction is therefore established. Hence, $\vec{\sigma}_s$ is an $\langle s, 0 \rangle$-route list, which, together with Step 4 and Theorem 2.2, implies that σ is feasible and $d(\sigma) \leq 2\mathrm{OPT}(\mathcal{I})$.

To see that the approximation ratio of 2 is tight, consider the instance $\mathcal{I}$ in Figure 2.2(a). Define $d(6, 8) := 1$ and $d(e) := 0$ for each $e \in E \setminus \{(6, 8)\}$. As we have shown, Algorithm 2.1 on $\mathcal{I}$ returns the same feasible route as that in Figure 2.2(b), whose total length equals 4. According to Figure 2.2(c), we have $\mathrm{OPT}(\mathcal{I}) = 2$. Thus, the approximation ratio of 2 is tight for Algorithm 2.1.

Moreover, for each leaf $u \in T$, by Lemma 2.3 the construction of $\vec{\sigma}_u$ runs in $O(n(u))$ time, which is in $O(1 + \sum_{v \in V} |q(v)|/k)$ time due to (2.1). For each internal node $u \in T$, by Lemma 2.4 the construction of $\vec{\sigma}_u$ runs in $O(n(l) + n(r))$ time, which is in $O(1 + \sum_{v \in V} |q(v)|/k)$ time due to (2.1). Hence, Algorithm 2.1 runs in $O(|V|(1 + \sum_{v \in V} |q(v)|/k))$ time. □

From the proofs of Theorem 2.2 and Theorem 2.3, we know that route σ returned by Algorithm 2.1 traverses each edge $e(v)$ with $v \in V$ at most $2n(v) + 2$ times. Thus, in the CTSPPD-T, although pickup and delivery requests can be split, σ serves each request by at most $n(v) + 1$ divisions, which is one time more than the minimum number of divisions for a full load service. Moreover, due to (2.1), we obtain that $\max_{v \in V} \mathrm{traversal}(\sigma, e(v))$ is $O(1 + \sum_{v \in V} |q(v)|/k)$, which, together with Theorem 2.1 and Theorem 2.3, implies that the tight approximation ratio of 2 can also be achieved in $O(|V|(1 + \sum_{v \in V} |q(v)|/k))$ time for any arbitrary instance. It is interesting to see that the demand-capacity ratio $\sum_{v \in V} |q(v)|/k$ arises as a key parameter in assessing the algorithm time complexity, since such a parameter also plays a key role in other similar problems (even if in the context of solution quality rather than algorithm time complexity) [5, 6]. Finally, although in this study we assume that each $q(v)$ is an integer, all the results obtained are valid even when each $q(v)$ is fractional. As a

result, the time complexity of the proposed algorithm is robust with respect to both rescaling of demands and the vehicle capacity.

2.6 Computational Results

This section reports on the computational results of the 2-approximation algorithm described in this chapter. For comparison, we also implemented a simple greedy algorithm that constructs a feasible route by starting from the root, alternating between pickup sub-routes and delivery sub-routes, and returning to the root after all the pickup and delivery requests are served, where each pickup sub-route keeps moving to the closest pickup point to pick up items until the vehicle's load is full, and each delivery sub-route keeps moving to the closest delivery point to deliver items until the vehicle's load is empty. It can be verified that the algorithm runs in $O(|V| \sum_{v \in V} |q(v)|/k + 2|V|^2)$ time. We coded both the algorithms in Java, and the experiments were performed on a Dell PC with a 2.83GHz CPU.

In our experiments we considered 1920 problem instances. For each $|V| \in \{20, 80, 320, 1280\}$, we generated 480 instances by first randomly generating ten trees in which vertices of each tree were sampled uniformly from a square $[0, 1] \times [0, 1]$ and connected randomly, each edge having a length equal to the distance of its endpoints. Each vertex v was assigned a product amount $q(v)$ randomly from $-\gamma$ to $+\gamma$, where γ is a parameter with its value chosen from $\{16, 32, 64, 128\}$. The vehicle capacity k was then chosen from $\{2^0, 2^1, 2^2, ..., 2^{11}\}$. Thus, for each value combination of γ, $|V|$, and k, we obtained 10 problem instances based on the ten trees.

According to the experiments, the 2-approximation algorithm exhibited much shorter running time than the greedy algorithm, particularly for large sized instances. For instances with $|V| = 1280$ and $k = 1$, the 2-approximation algorithm took less than 1 minute on average, including the time of the transformation to standard instances, while the greedy algorithm took more than 40 minutes, which is greatly slower

than the 2-approximation algorithm. The main reason is that, for these randomly generated instances, when $k = 1$, the expected time complexity of the greedy algorithm is $O(|V| \sum_{v \in V} |q(v)|)$ but the 2-approximation algorithm is only $O(\sum_{v \in V} |q(v)|)$. Due to evenly distributed pickup and delivery points and evenly generated product amount in these randomly instances, the internal node $u \in V$ can have an expected total balance equals to zero, i.e., $q(T_u) = 0$. According to (2.1), $n(u) = 1$ holds for each internal node u, leading to the improvement on time complexity of the 2-approximation algorithm.

Table 2.2: Average approximation ratios of the 2-approximation algorithm (A) and the greedy algorithm (G) over randomly generated instances.

γ	k	k/γ	$\|V\|$=20		$\|V\|$=80		$\|V\|$=320		$\|V\|$=1280	
			A	G	A	G	A	G	A	G
16	1	0.06	1.00	1.06	1.13	1.19	1.00	1.06	1.00	1.07
16	2	0.13	1.02	1.11	1.20	1.29	1.03	1.12	1.03	1.14
16	4	0.25	1.07	1.19	1.09	1.22	1.09	1.23	1.09	1.26
16	8	0.50	1.17	1.27	1.19	1.36	1.19	1.41	1.20	1.45
16	16	1.00	1.19	1.33	1.32	1.55	1.32	1.63	1.35	1.70
16	32	2.00	1.08	1.19	1.28	1.48	1.31	1.69	1.32	1.72
16	64	4.00	1.00	1.19	1.15	1.42	1.22	1.56	1.22	1.65
16	128	8.00	1.00	1.00	1.05	1.16	1.12	1.45	1.13	1.55
16	256	16.00	1.00	1.00	1.00	1.18	1.02	1.42	1.05	1.49
16	512	32.00	1.00	1.00	1.00	1.00	1.00	1.25	1.00	1.41
16	1024	64.00	1.00	1.00	1.00	1.00	1.00	1.19	1.00	1.39
16	2048	128.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.22
32	1	0.03	1.00	1.05	1.00	1.07	1.00	1.06	1.00	1.07
32	2	0.06	1.01	1.07	1.02	1.11	1.02	1.10	1.02	1.10
32	4	0.13	1.05	1.13	1.05	1.17	1.05	1.16	1.05	1.17
32	8	0.25	1.09	1.22	1.10	1.29	1.11	1.28	1.10	1.29
32	16	0.50	1.20	1.33	1.20	1.45	1.22	1.47	1.21	1.47
32	32	1.00	1.32	1.43	1.37	1.71	1.38	1.73	1.36	1.72
32	64	2.00	1.18	1.38	1.32	1.61	1.35	1.72	1.32	1.73
32	128	4.00	1.03	1.27	1.20	1.48	1.23	1.59	1.23	1.65
32	256	8.00	1.00	1.04	1.08	1.29	1.14	1.49	1.11	1.57
32	512	16.00	1.00	1.00	1.00	1.16	1.03	1.44	1.05	1.49

continued on next page

continued from previous page

γ	k	k/γ	$\|V\|$=20		$\|V\|$=80		$\|V\|$=320		$\|V\|$=1280	
			A	G	A	G	A	G	A	G
32	1024	32.00	1.00	1.00	1.00	1.00	1.01	1.25	1.01	1.43
32	2048	64.00	1.00	1.00	1.00	1.00	1.00	1.24	1.00	1.41
64	1	0.02	1.00	1.04	1.00	1.04	1.00	1.06	1.00	1.06
64	2	0.03	1.01	1.05	1.01	1.06	1.01	1.08	1.01	1.08
64	4	0.06	1.02	1.08	1.02	1.09	1.03	1.12	1.03	1.11
64	8	0.13	1.06	1.12	1.05	1.15	1.06	1.18	1.06	1.18
64	16	0.25	1.10	1.19	1.10	1.26	1.12	1.30	1.12	1.30
64	32	0.50	1.18	1.33	1.20	1.41	1.23	1.48	1.23	1.50
64	64	1.00	1.32	1.47	1.34	1.64	1.40	1.76	1.39	1.76
64	128	2.00	1.23	1.34	1.31	1.57	1.36	1.73	1.35	1.77
64	256	4.00	1.09	1.17	1.20	1.49	1.21	1.61	1.24	1.67
64	512	8.00	1.00	1.02	1.02	1.35	1.11	1.49	1.14	1.58
64	1024	16.00	1.00	1.00	1.00	1.17	1.02	1.43	1.06	1.51
64	2048	32.00	1.00	1.00	1.00	1.00	1.00	1.24	1.01	1.46
128	1	0.01	1.00	1.04	1.00	1.05	1.00	1.05	1.00	1.06
128	2	0.02	1.00	1.05	1.00	1.05	1.00	1.06	1.00	1.07
128	4	0.03	1.01	1.06	1.01	1.07	1.01	1.08	1.01	1.09
128	8	0.06	1.02	1.09	1.02	1.10	1.03	1.12	1.03	1.13
128	16	0.13	1.05	1.12	1.06	1.16	1.06	1.18	1.06	1.20
128	32	0.25	1.10	1.19	1.12	1.27	1.12	1.30	1.13	1.32
128	64	0.50	1.19	1.31	1.22	1.42	1.23	1.50	1.24	1.51
128	128	1.00	1.34	1.52	1.34	1.65	1.40	1.73	1.40	1.78
128	256	2.00	1.25	1.26	1.33	1.61	1.33	1.73	1.34	1.76
128	512	4.00	1.05	1.13	1.24	1.38	1.22	1.60	1.24	1.67
128	1024	8.00	1.00	1.00	1.08	1.31	1.10	1.49	1.13	1.57
128	2048	16.00	1.00	1.00	1.00	1.20	1.01	1.43	1.07	1.49

We measured the quality of a solution by the approximation ratio between the length of the solution and the lower bound on the length of the optimal solution shown in the left hand side of (2.2). Table 2.2 reports the average approximation ratios for both the 2-approximation algorithm and the greedy algorithm over all 10 instances for each value combination of γ, $|V|$, and k. It can be seen that the average approximation ratios are sensitive to k/γ and $|V|$. When k/γ is either extremely small or extremely large, the average approximation ratio of the 2-approximation algorithm is close to 1. When k/γ approaches 1, the ratio reaches the largest value,

which never exceeds 1.40. When $|V|$ increases from 20, 80, 320, to 1280, the average ratio increases when $k/\gamma \geq 1$, but changes little when $k/\gamma < 1$. The results also show that the 2-approximation algorithm performed significantly better than the simple greedy algorithm. For instances with $k/\gamma = 1$ and $|V| = 1280$, the average approximation ratio of the greedy algorithm is 1.74, more than 26.5% larger than that of the 2-approximation algorithm, whose average ratio is 1.375.

2.7 Summary

In this chapter, we have developed a 2-approximation algorithm for the CTSPPD-T. The new improved algorithm extends an exact algorithm for the CTSPPD-P from the literature and employs more complicated techniques to construct the solution. Our algorithm has a polynomial time complexity under a reasonable and practical condition, which is similar with that in the existing literature. Comparing with the existing approximation algorithms derived from the CTSPPD literature, our algorithms not only improves on the solution quality, i.e., with a better approximation ratio, but also improves on the time complexities, i.e., under a more relax condition. Moreover, computational results show that our algorithm also achieves good average performance over randomly generated instances and exhibits a much shorter running time and better solution quality than a greedy algorithm. According to the assumptions we have made in this chapter, our future work will mainly focus on whether the unbalanced case still has constant ratio approximation algorithms and whether the results in this chapter can be further extended to a more complicated setting for the problem.

CHAPTER 3

Improved Algorithms for Joint Optimization of Facility Locations and Network Connections

3.1 Introduction

Consider a complete undirected graph $G = (V, E)$ where $V = \{1, 2, ..., n\}$ denotes the vertex set, E denotes the edge set, and each edge $e \in E$ has a non-negative weight denoted by $\ell(e)$. Let $W \subseteq V$ denote a set of potential facilities, and $J \subseteq V$ a set of clients, where both W and J are not empty. Let k with $1 \leq k \leq |W|$ indicate the maximum number of facilities that are allowed to be opened. Each client in J needs to be served by connecting it to an open facility along a path. The resulting connections can be represented by *a k-median Steiner forest*, which is defined as a collection of at most k trees that covers all the clients, with each tree containing a distinct facility location as the root, where vertices in $V \setminus J$ are Steiner vertices that may or may not be included in the forest. To minimize the total connection cost, we study in this chapter *a k-median Steiner forest problem* that aims to find an *optimal k-median Steiner forest* that minimizes the total edge weight. See Figure 3.1.

The k-median Steiner forest problem aims to jointly optimize facility locations and network connections. This problem has wide applications, particularly in the telecommunication and transportation industries, where facilities, such as service centers or

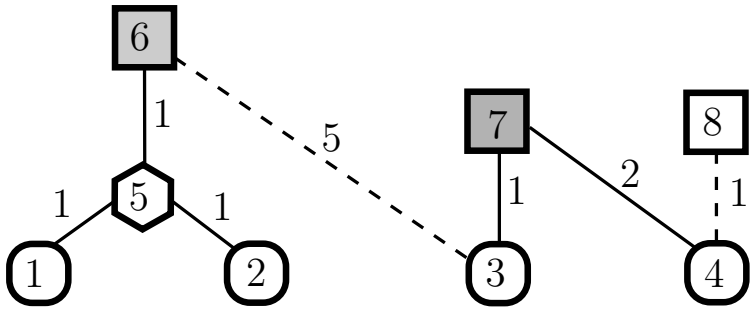


Figure 3.1: An instance of the k-median Steiner forest problem with $k = 2$, $V = \{1, 2, ..., 8\}$, $J = \{1, 2, 3, 4\}$ and $W = \{6, 7, 8\}$: Vertices 5, 6, 7, and 8 are Steiner vertices; the numbers on the edges indicate edge weights; for each edge not shown, its weight equals the total edge weight of the shortest path that connects its endpoints; the optimal k-median Steiner forest (shown in solid lines) opens facilities 6 and 7, and has a total edge weight of 6.

factories, need to be located and connected to clients by cable or road constructions. Its solutions can also be utilized to construct plans of facility locations combined with vehicle routing or cargo shipping [16, 61, 64].

The k-median Steiner forest problem is strongly $\mathcal{NP}$-hard, since it contains the classical Steiner tree problem as a special case with $|W| = k = 1$. As with the classical Steiner tree problem [73], it can be assumed without loss of generality that the weight of each edge equals the total edge weight of the shortest path that connects the endpoints of the edge, so that the edge weights satisfy triangle inequality and form a *metric*. This is because each edge of a k-median Steiner forest can always be replaced by the shortest path that connects its endpoints, without increasing the total edge weight of the Steiner forest.

Since the k-median Steiner forest problem is strongly $\mathcal{NP}$-hard, it is of great interest to develop approximation algorithms for it with provable guarantees on running time and solution quality, as well as to identify special cases that are commonly seen in practice and can be solved to optimality by polynomial time algorithms. In this chapter, we develop an improved approximation algorithm for the k-median Steiner forest problem, as well as new polynomial time algorithms that can solve two non-

trivial special cases of the problem.

3.1.1 Previous Work

For the k-median Steiner forest problem, only a 2-approximation algorithm is known in the literature [64, 16]. It is based on a Lagrangian relaxation of the problem, and uses a sophisticated primal-dual schema that holds a so-called Lagrangian preserving performance guarantee to construct Steiner forests. (See a detailed review in Section 3.2.3.1.) As a result, the proof of its approximation ratio is complicated.

The k-median Steiner forest problem is a joint optimization problem that takes into account decisions on both facility locations and network design [27]. Among many facility location problems that have been extensively studied in the literature, the k-median problem is the most relevant one, which aims to open at most k facilities and *directly* connect each client to an open facility by an edge with the total edge weight minimized. For the k-median problem, [20] achieved the first constant approximation ratio of 6.67, and recently, based on a breakthrough made by [53], [15] achieved the current best approximation ratio of $2.61 + \epsilon$ for any $\epsilon > 0$. The existing 2-approximation algorithm for the k-median Steiner forest problem [16] followed the approach of a 6-approximation algorithm developed by [42] for the k-median problem. [42] reduced the k-median problem to an *uncapacitated facility location problem* by relaxing the constraint of opening at most k facilities, and penalizing the opening of each facility by a Lagrangian multiplier. They then applied a primal-dual schema to obtain a 3-approximation of the uncapacitated facility location problem, and transformed it to a 6-approximation of the k-median problem. This approach has been improved by [40] and [8] to achieve approximation ratios of 4 and $3 + \epsilon$ for the k-median problem.

Among various network design problems that have been extensively studied in the literature, the classical Steiner tree problem is the most relevant one, which,

as mentioned earlier, is a special case of the k-median Steiner forest problem with $|W| = k = 1$, aiming to minimize the total cost of connecting all the clients to a given facility. It is well-known that a minimum spanning tree of the subgraph induced by the clients can lead to a 2-approximation of the Steiner tree problem [73]. Moreover, approximation ratios smaller than 2 have been achieved by [78, 44, 62, 67], and with the current best approximation ratio being $\ln(4) + \epsilon < 1.39$, recently achieved by [14]. Moreover, [63] proposed a primal-dual schema that achieves an approximation ratio of 2 for a Steiner forest problem, which aims to connect clients by a given number of trees with the total edge weight minimized. This problem is equivalent to a special case of the k-median Steiner forest problem, where each vertex contains a facility, i.e., $W = V$.

Solutions to the k-median Steiner forest problem are often used to construct approximations of other problems that jointly optimize facility locations and network design. [64] studied a location-shipping problem that aims to open at most k facilities and install cables of sufficient capacity for shipping cargo from clients to facilities. By combining the 2-approximation of the k-median Steiner forest problem and a ρ-approximation of the k-median problem, they constructed a $(\rho + 2, 2)$ bicriteria approximation with a total cost at most $\rho + 2$ times that of the optimal solution, and with a total of at most $2k$ facilities opened. [16] studied a k-location-routing problem that aims to assign depots (facilities) to k vehicles and to route the vehicles to serve clients, with the total routing cost minimized. By duplicating each edge of the 2-approximation of the k-median Steiner forest problem, they obtained a collection of k tours with each tour starting and ending at an open facility, and proved that such a tour collection is a 2-approximation of the k-location-routing problem. Moreover, [76] studied a special case of the k-location-routing problem, where vertices are located in a tree-shaped network and edge weights represent lengths of shortest paths on the tree. Such a tree shaped network appears in several manufacturing and logistics

applications [10, 22, 24, 45, 74], including those in rural or water transportation systems [72, 77]. For this special case, solutions to the k-median Steiner forest problem and solutions to the k-location-routing problem are one-to-one correspondence, and therefore these two problems are equivalent. [76] developed an algorithm that can solve this special case of the problem to optimality in $O(n2^{6k})$ time. However, when k is part of the input, whether or not this special case has a polynomial time algorithm still remains open.

3.1.2 Our Results

For the k-median Steiner forest problem, we have developed a new 2-approximation algorithm. It is simpler than the existing 2-approximation algorithm of [16], consisting of only an $O(n^2)$-time transformation from a minimum spanning tree of the clients and a new vertex that replaces all the facilities. This extends the well-known result that a minimum spanning tree of the clients can lead to a 2-approximation of the Steiner tree problem. Compared with the existing 2-approximation algorithm, our new algorithm has an improved approximation ratio of $2 - 1/|J|$, which is not only tight but easier to be proved, and it can always produce solutions of equal or better quality, the quality improvement being up to 50% in some cases.

Moreover, we have developed new polynomial time algorithms that can solve two non-trivial special cases of the k-median Steiner forest problem to optimality. For a special case where each vertex contains a client, i.e, $J = V$, we show that it is equivalent to a problem of finding a minimum weighted basis for a matroid, and that its optimal solution can be obtained in polynomial time by a transformation from a minimum spanning tree of the clients. This result is interesting because the same special case for the k-median problem, where $J = V$, is still strongly $\mathcal{NP}$-hard. For the other special case, where vertices are located in a tree-shaped network, we develop a dynamic programming algorithm that, for the first time in the literature, can solve

the problem to optimality in polynomial time, a significant improvement over the existing best algorithm of $O(n2^{6k})$ running time [76].

The remainder of this chapter is organized as follows: In Section 3.2 we present the improved approximation algorithm for the k-median Steiner forest problem, and compare its performance with the existing 2-approximation algorithm. In Section 3.3 we present the new polynomial time algorithms for the two special cases of the problem. We then summarize the chapter in Section 3.4 with discussions on applications of the results obtained as well as directions for future research.

3.2 Improved Approximation Algorithm

We present our new approximation algorithm for the k-median Steiner forest problem in Section 3.2.1, prove its tight approximation ratio of $2 - 1/|J|$ in Section 3.2.2, and then show its improvement over the existing 2-approximation algorithm in Section 3.2.3.

As mentioned earlier, we can assume without loss of generality that edge weights satisfy triangle inequality. Throughout this section, we also assume that W and J are disjoint, since each vertex that belongs to both W and J can be duplicated to two vertices, one being a facility and the other being a client.

3.2.1 The new algorithm

Our new approximation algorithm consists of the following three steps:

Step 1. Construct a complete undirected graph H by shrinking vertices of W to a new vertex r and eliminating vertices not in J. As a result, H is a complete graph on vertices of $J \cup \{r\}$. For each edge (u, v) with $u \in J$ and $v \in J$, its weight still equals $\ell(u, v)$, and for each edge (r, v) with $v \in J$, we let $\ell(r, v) = \min\{\ell(u, v) : u \in W\}$.

Step 2. Define a *k-root-degree spanning tree* of H, or k-RDST of H in short, as a spanning tree of H, with r being its root, such that the degree of r does not exceed k. Compute a *minimum k-RDST* T_H, which is a k-RDST of the minimum total edge weight.

Step 3. Construct from T_H a k-median forest F of G with $\ell(F) = \ell(T_H)$, as follows. For each edge (r, v) of T_H with $v \in J$, we replace it with an edge (u, v) where $u \in W$ and $\ell(u, v) = \ell(r, v)$. Since the degree of r in T_H does not exceed k, we know that the resulting tree collection F must contain at most k trees, with each tree containing a distinct facility in W. Thus, F is a k-median Steiner forest of G. Since each edge (r, v) of T_H with $v \in J$ corresponds to an edge (u, v) with $u \in W$ of equal weight, we have $\ell(F) = \ell(T_H)$. Since $k \leq n$, this step takes $O(n)$ time.

It can be seen that the above algorithm constructs a k-median Steiner forest F of G by an $O(n)$-time transformation from a minimum k-RDST T_H of H with $\ell(F) = \ell(T_H)$, where H is a complete graph with its vertices including all the clients, as well as an additional vertex r that replaces all the facilities. We next show that T_H can be computed by an $O(n^2)$-time transformation from a minimum spanning tree of H.

Lemma 3.1. *For any minimum spanning tree T^* of H, it can be transformed to a minimum k-RDST T_H of H in $O(n^2)$-time.*

Proof. For $1 \leq q \leq k$, let $T(q)$ indicate a spanning tree of H that minimizes the total edge weight, with the degree of r equal to q. Thus, among all $T(q)$ for $1 \leq q \leq k$, the one with the minimal total edge weight is a minimum k-RDST of H. It is known from [31] that each $T(q)$ can be obtained by an $O(n^2)$-time transformation from the minimum spanning tree T^* of H. Thus, a minimum k-RDST of H can be obtained by applying k times such a transformation. Next, we show that it is sufficient to

apply at most once such a transformation. To see this, consider the following two cases: If the degree of r in T^* does not exceed k, then we directly obtain that T^* is a minimum k-RDST of H. Otherwise, the degree of r in T exceeds k. For this case, it is known from [56] that there exists a minimum k-RDST of H with the degree of r exactly equal to k. Thus, $T(k)$ is a minimum k-RDST of H. Since $T(k)$ can be obtained by an $O(n^2)$-time transformation from T^* [31], Lemma 3.1 is proved. □

By Lemma 3.1, the new approximation algorithm is also an $O(n^2)$-time transformation from a minimum spanning tree of H. It is noted that the fastest minimum spanning tree algorithm runs in $O(\alpha(|E|, \sqrt{|E|})|E|)$ time, where $\alpha(|E|, \sqrt{|E|})$ is the functional inverse of Ackermann's function, which grows very slowly and can be considered as a constant [21]. Since $|E|$ is in $O(n^2)$, we obtain that our new algorithm runs in $O(\alpha(n^2, n)n^2)$ time.

Example 3.1. Apply the new algorithm on the instance shown in Figure 3.1, where $k = 2$. In H constructed by Step 1, we have $\ell(r, v) = 1$ for $v \in \{3, 4\}$, $\ell(r, v) = 2$ for $v \in \{1, 2\}$, $\ell(1, 2) = 2$, and $\ell(3, 4) = 3$. By the definition in Step 2, it is easy to verify that the tree $(\{r, 1, 2, 3, 4\}, \{(r, 1), (1, 2), (r, 3), (3, 4)\})$ is a minimum k-RDST T_H of H with $\ell(T_H) = 8$. Thus, replacing edges $(r, 1)$ and $(r, 3)$ with $(6, 1)$ and $(7, 3)$, we obtain a k-median Steiner forest F of G in Step 3 of $\ell(F) = \ell(T_H) = 8$.

3.2.2 Approximation ratio of the new algorithm

Consider the k-median Steiner forest F obtained by the new approximation algorithm. Let OPT indicate an optimal k-median Steiner forest. We prove as follows that the new approximation algorithm guarantees an approximation ratio of $2-1/|J|$.

Theorem 3.1. $\ell(F) \leq (2 - 1/|J|)\ell(\text{OPT})$.

Proof. For each tree T of OPT, let w_T indicate its facility, n_T indicate the number of clients in T, and P_T indicate the path of the largest total edge weight that connects

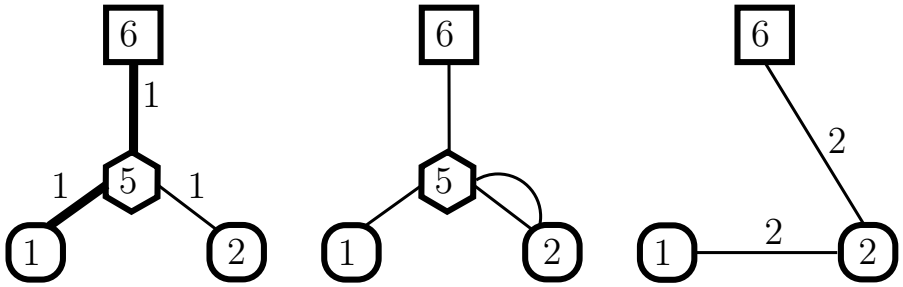
w_T and a client in T, for which we use $v_T \in J$ to indicate the endpoint of P_T that is a client. We duplicate each edge of T except those on P_T, so as to obtain a subgraph that contains an Eulerian path that starts from w_T, visits every edge exactly once, and then returns to v_T. Thus, by short-cutting vertices not in $J \cup W$ as well as facilities other than w_T at the start, we obtain a path T' that starts from w_T and visits all the clients of T exactly once. See Figure 3.2.

Hence, by repeating the above procedure for every tree T in OPT, and replacing all the facilities with r, we can obtain a collection of at most k paths that all start from r, cover all the clients, and contain no other vertices. Thus, it forms a k-RDST of H, denoted by T'_H, and we have $\ell(T_H) \leq \ell(T'_H)$. Hence, by $\ell(F) = \ell(T_H)$, we have $\ell(F) \leq \ell(T'_H)$.

Moreover, since OPT is an optimal k-median Steiner forest, without loss of generality it can be assumed that each leaf of the trees in OPT is a client. Thus, for each tree $T \in \text{OPT}$, since every edge of T belongs to a certain path that connects w_T and a client in T, we obtain that $\ell(P_T) \geq \ell(T)/n_T$. By triangle inequality, we know $\ell(T'_H) \leq 2[\ell(\text{OPT}) - \sum_{T\in\text{OPT}} \ell(P_T)] + \sum_{T\in\text{OPT}} \ell(P_T) = 2\ell(\text{OPT}) - \sum_{T\in\text{OPT}} \ell(P_T)$, which, together with $\ell(P_T) \geq \ell(T)/n_T$ and $n_T \leq |J|$ for each $T \in \text{OPT}$, implies that $\ell(T'_H) \leq (2 - 1/|J|)\ell(\text{OPT})$. Hence, we obtain that $\ell(F) \leq \ell(T'_H) \leq (2 - 1/|J|)\ell(\text{OPT})$. □

Moreover, the approximation ratio $(2 - 1/|J|)$ is tight by the following example.

Example 3.2. Let $k = 1$, $n = h + 2$, $J = \{1, 2, .., h\}$, and $W = \{h + 1, h + 2\}$. Define $\ell(h + 1, u) = 1$ for $u \in J$, and $\ell(h + 2, 1) = 1$. For other edges (u, v), define $\ell(u, v) = 2$. It can be seen that the edge weights satisfy triangle inequality, and that it is optimal to connect facility $h+1$ to each client $u \in J$ by an edge $(h+1, u)$, so that $\ell(\text{OPT}) = h = |J|$. Moreover, since edges $(h + 2, 1)$ and $(1, v)$ for $2 \leq v \leq h$ form a minimum k-RDST of H, it can be seen that the approximation solution F produced



(a) T, where $P_T = \langle 6, 5, 1 \rangle$ is shown in bold lines

(b) After duplicating each edge of T except those on P_T

(c) T' with $\ell(T') = 4$

Figure 3.2: Illustration of the construction of T' from T in the proof of Theorem 3.1, where T is one of the trees of the optimal solution for the instance shown in Figure 3.1.

by the proposed algorithm consists of edges $(h+2, 1)$ and $(1, v)$ for $2 \leq v \leq h$, so that $\ell(F) = 1 + 2(h-1) = 2|J| - 1 = (2 - 1/|J|)|J| = (2 - 1/|J|)\ell(\mathrm{OPT})$. Thus, the approximation ratio of $(2 - 1/|J|)$ is tight for the proposed approximation algorithm.

3.2.3 Comparison with the existing 2-approximation algorithm

To compare our new algorithm with the existing 2-approximation algorithm developed by [16], we first provide a detailed review of the existing algorithm in Section 3.2.3.1, and prove its equivalence to a transformation from an optimal solution of a minimum spanning tree problem in Section 3.2.3.2. Based on this equivalence, we then show the improvements made by our new algorithm in Section 3.2.3.3.

3.2.3.1 Review of the existing 2-approximation algorithm

[16] studied a problem that is equivalent to the k-median Steiner forest problem. It can be formulated as a Steiner tree problem on a new graph $\hat{G}$ with a root degree constraint, where the vertex set of $\hat{G}$ consists of V and a new root vertex r, and the edge set of $\hat{G}$ consists of E and root edges (r, v) for $v \in W$ with $\ell(r, v) = 0$. It can be seen that the k-median Steiner forest problem is equivalent to finding a Steiner tree of $\hat{G}$ of minimum total edge weight, such that it connects the root r and all clients

$v \in J$ with the degree of r not exceeding k. As a result, a facility $v \in W$ is open if, and only if, the Steiner tree of $\hat{G}$ contains the root edge (r, v).

To construct such a Steiner tree of $\hat{G}$, [16] first relaxed the degree constraint on r and introduced a non-negative Lagrangian multiplier λ to penalize selections of root edges (r, v) for $v \in W$. For any given $\lambda \geq 0$, they proposed a primal-dual schema to construct a Steiner tree of $\hat{G}$ such that the sum of its total edge weight and its total penalty do not exceed twice the sum of the total edge weight and total penalty of the optimal solution OPT. They then proved that there exists a value λ_k such that the Steiner tree obtained for $\lambda = \lambda_k$ is a 2-approximation of the problem.

According to [16], the primal and dual schema that constructs the Steiner tree of $\hat{G}$ for a given λ is equivalent to applying a truncated version of the well-known Kruskal's minimum spanning tree algorithm on $\hat{G}$ under revised edge weights $\ell_\lambda(\cdot)$, where $\ell_\lambda(u, v) = \ell(u, v)/2$ for $u \in J$ and $v \in J$, $\ell_\lambda(u, v) = \ell(u, v)$ for $u \in W$ and $v \in J$, $\ell_\lambda(r, u) = s(u) + \lambda$ for $u \in W$, where $s(u)$ indicates the smallest value of $\ell(u, v)$ with $v \in J$, and for other edges (u, v), it sets $\ell_\lambda(u, v) = \infty$. See Figure 3.3 for an example of $\ell_\lambda(\cdot)$. The primal and dual schema starts with an empty set of edges, and iteratively selects and adds an edge (u, v) of the smallest weight such that v belongs to an active connected component C_v that does not contain u. Here, an active connected component is defined as a connected component of the subgraph induced by the selected edges such that the component contains at least one client in J, but does not contain the root r. Among all such edges of the same length, ties are broken (i) by preferring root edges if less than k root edges have been selected, and preferring non-root edges otherwise, and (ii) by not selecting edges that join a facility and a client if other choices are available. Edges (u, v) with $u \in W$ and $v \in W$ are never selected, since $\ell_\lambda(u, v) = \infty$. When no new edges can be selected, the iteration stops and a Steiner tree $\hat{T}_\lambda$ of $\hat{G}$ that connects r and all clients in J is obtained.

[16] proved the following lemma for the resulting Steiner tree $\hat{T}_\lambda$.

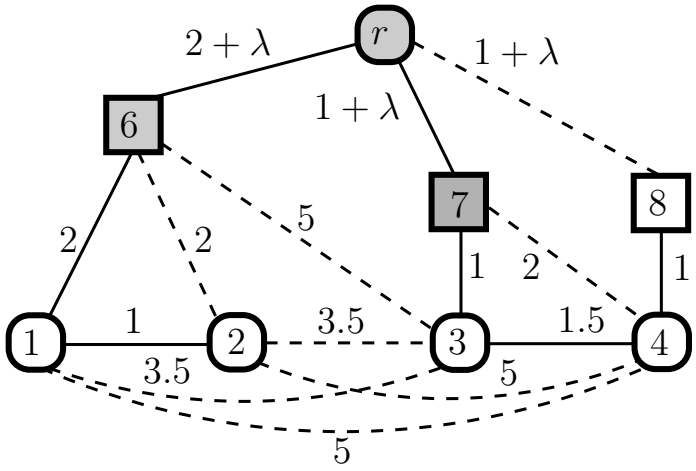


Figure 3.3: Illustration of $\hat{G}$ for the instance in Figure 3.1: The numbers on the edges indicate the edge weights under $\ell_\lambda(\cdot)$; for edges not shown, their weights under $\ell_\lambda(\cdot)$ are too large to be selected, and are omitted; T_{λ_k} with $\lambda_k = 1$ are shown in solid lines.

Lemma 3.2 ([16]). *For each $\lambda \geq 0$, in $\hat{T}_\lambda$ every facility $u \in W$ is incident with at most one edge (u, v) with $v \in J$, and if $\hat{T}_\lambda$ contains an edge (u, v) for $u \in W$ and $v \in J$, then $\ell(u, v) = s(u)$.*

Proof. The proof is presented below for completeness. For each $\lambda \geq 0$, consider the Steiner tree $\hat{T}_\lambda$ of $\hat{G}$ constructed above.First, if in $\hat{T}_\lambda$, a facility $u \in W$ is incident with two edges (u, v) and (u, v') with $v \in J$ and $v' \in J$, then by triangle inequality for $\ell(\cdot)$ and by the definition of $\ell_\lambda(\cdot)$ in Section 3.2.3.1, we know that $\ell_\lambda(v, v') = \ell(v, v')/2 \leq [\ell(u, v) + \ell(u, v')]/2$, implying that either $\ell_\lambda(v, v') \leq \ell(u, v) = \ell_\lambda(u, v)$ or $\ell_\lambda(v, v') \leq \ell(u, v') = \ell_\lambda(u, v')$. Thus, due to the preference adopted for tie breaking in the construction of T_λ, we know that $\hat{T}_\lambda$ cannot contain both (u, v) and (u, v'). Thus, every facility $u \in W$ is incident with at most one edge (u, v) with $v \in J$.

Next, if $\hat{T}_\lambda$ contains (u, v) for $u \in W$ and $v \in J$, then by the argument above we know that at the time when (u, v) is added to $\hat{T}_\lambda$, no edge (u, v') with $v' \in J$ is selected, and thus $\ell_\lambda(u, v)$ cannot be larger than $\ell_\lambda(u, v')$ for any other (u, v') with $v' \in J$ and $v' \neq v$, which, together with $\ell_\lambda(u, v) = \ell(u, v)$ and $\ell_\lambda(u, v') = \ell(u, v')$, implies that $\ell(u, v) = s(u)$. This completes the proof of Lemma 3.2. □

According to Lemma 3.2, one can eliminate every edge (u, v) with $u \in W$, $v \in J$, and (r, u) not in $\hat{T}_\lambda$, such that the remainder is still a Steiner tree of $\hat{G}$, denoted by T_λ. Let $\delta(r) = \{(r, v) : v \in W\}$ denote the set of root edges. Let x_e be a binary variable that indicates whether or not T_λ contains an edge $e \in E \cup \delta(r)$. [16] proved that T_λ has a Lagrangian preserving performance guarantee, as shown below:

$$\sum_{e \in E} \ell(e) x_e + 2 \sum_{e \in \delta(r)} \lambda x_e \leq 2\ell(\text{OPT}) + 2k\lambda. \tag{3.1}$$

Next, [16] showed as follows that there exists $\lambda_k \geq 0$ such that T_{λ_k} is a 2-approximation of the problem. If T_0 (for $\lambda = 0$) contains no more than k edges of $\delta(r)$, then by (3.1) we have $\ell(T_0) \leq 2\ell(\text{OPT})$, and thus T_0 is a 2-approximation of the problem, so that one can set $\lambda_k = 0$. Otherwise, T_0 contains more than k edges of $\delta(r)$. For this case, [16] proved that there exists $\lambda_k > 0$ such that T_{λ_k} contains exactly k edges of $\delta(r)$. The computation of λ_k depends on how edges are selected through the construction of T_∞ (for $\lambda = \infty$), in which only one edge of $\delta(r)$ is selected. For each edge (u, v) of T_∞ with $u \in J$ and $v \in J$, consider the time before (u, v) is selected. Let C_u and C_v indicate the connected components that contain u and v, respectively, in the subgraph induced by the edges selected before (u, v). Let $s(C_u)$ and $s(C_v)$ indicate the smallest values of $s(i)$ for all facilities i in C_u and C_v, respectively. Define $s(u, v) := \max\{s(C_u), s(C_v)\}$. It can be seen that if $\ell(u, v)$ exceeds $s(u, v) + \lambda$, then both C_u and C_v must become inactive before (u, v) can be selected, and thus (u, v) will not be selected. In light of this, [16] defined λ_k as the value of λ such that exactly $|J| - k$ edges of T_∞ with $u \in J$ and $v \in J$ are selected in T_λ for $\lambda = \lambda_k$. Thus, by Lemma 3.2, it can be seen that T_{λ_k} must contain exactly k edges of $\delta(r)$ to form a Steiner tree of $\hat{G}$. By (3.1), we have $\ell(T_{\lambda_k}) \leq 2\ell(\text{OPT})$, implying that T_{λ_k} is a 2-approximation of the problem.

Example 3.3. Apply the algorithm of [16] on the instance in Figure 3.1, where $\hat{G}$

is shown in Figure 3.3. When $\lambda = 0$, it can be seen that edges $(1,2)$, $(7,3)$, $(r,7)$, $(8,4)$, $(r,8)$, $(6,1)$ and $(r,6)$ are selected one by one to form T_0, which contains all the three edges (i.e., more than $k = 2$ edges) in $\delta(r) = \{(r,6),(r,7),(r,8)\}$. Next, when $\lambda = \infty$, it can be seen that edges $(1,2)$, $(7,3)$, $(8,4)$, $(3,4)$, $(6,1)$, $(2,3)$, and $(r,7)$ are selected one by one to form T_∞. From this, we can see that when $\lambda = 1$, edges $(1,2)$, $(7,3)$, $(8,4)$, and $(3,4)$ are still selected one by one in the construction of T_1. Since $\ell_1(r,7) = 1+1 = 2 = \ell_1(6,1)$, edges $(r,7)$ and $(6,1)$ are then included in T_1. Now, consdier edge $(2,3)$, which connects connected components C_2 of vertices in $\{1,2,6\}$ and C_3 of vertices in $\{3,7,r\}$, implying that $s(2,3)+\lambda = \max\{2+\lambda, 1+\lambda\} = 3 < 3.5 = \ell_1(2,3)$. Thus, edge $(2,3)$ is not include in T_1. Moreover, since $\ell_1(r,6) = 3$, edge $(r,6)$ is then selected in T_1. Hence, T_1 consists of edges in $\{(1,2),(7,3),(8,4),(3,4),(r,7),(6,1),(r,6)\}$, including exactly $|J| - k = 2$ edges (u,v) of T_∞ with $u \in J$ and $v \in J$. Thus, we can set $\lambda_k = 1$. By removing $(8,4)$ (since $(r,8)$ is not selected), and removing $(r,6)$ and $(r,7)$, we obtain from T_1 a k-median Steiner forest of G.

We have seen that the existing 2-approximation algorithm, developed by [16], is a primal-dual schema that is equivalent to a truncated version of the well-known Kruskal's minimum spanning tree algorithm, for which the best implementation takes $O(|E|\log n)$ or $O(n^2 \log n)$ time, slower than our newly proposed approximation algorithm by a factor of $\log n$. Moreover, the proof of its approximation ratio relies on (3.1), which is derived from a complicated construction and analysis of a dual feasible solution associated with the primal-dual schema. Compared with this, the proof of the approximation ratio in Theorem 3.1 for our algorithm is much simpler.

3.2.3.2 A simple equivalence of the existing 2-approximation algorithm

We now show that the existing 2-approximation algorithm, developed by [16], is equivalent to a transformation from an optimal solution to a minimum spanning tree

problem. For this, by Lemma 3.1, it is sufficient for us to show as follows that it is equivalent to a transformation from a minimum k-RDST of H under some edge weights that are different from $\ell(\cdot)$, where H, as defined earlier in Section 3.2.1, is a complete graph on $J \cup \{r\}$.

For any $\lambda \geq 0$, we are reminded that edges (u, v) with $u \in W$ and $v \in W$ are never selected in T_λ due to $\ell_\lambda(u, v) = \infty$. Thus, by Lemma 3.2 we know that for each facility $u \in W$ of T_λ, it must be incident and only incident with two edges in T_λ, including the root edge (r, u) and the other edge (u, v) with $v \in J$ and $\ell(u, v) = s(u)$. Thus, we can transform T_λ to a spanning tree S_λ of H by replacing (r, u) and (u, v) with (r, v). Consider revised edge weights $\pi_\lambda(\cdot)$ for edges of H, where $\pi_\lambda(r, v) = \min\{\ell(u, v) : u \in W\} + \lambda$ for $v \in J$, and $\pi_\lambda(u, v) = \ell(u, v)/2$ for $u \in J$ and $v \in J$. We can then establish the following lemma for S_λ.

Lemma 3.3. *S_λ is a minimum spanning tree of H under edge weights $\pi_\lambda(\cdot)$, for $\lambda \geq 0$.*

Proof. Note that S_λ is obtained from T_λ, which is obtained from $\hat{T}_\lambda$. From the construction of $\hat{T}_\lambda$ we know that S_λ can also be obtained in the following way: Start with an empty edge set for S_λ. For each edge (u, v) of $\hat{T}_\lambda$ according to their sequence in the construction of $\hat{T}_\lambda$, if $u \in J$ and $v \in J$, then we add (u, v) to S_λ; if $u \in W$ and $v \in J$, we ignore (u, v) (since it does not appear in S_λ); and if $u = r$ and $v \in W$, then we add (r, i) to S_λ with $i \in J$ and $\ell(v, i) = s(v)$. By Lemma 3.2, it can then be seen that S_λ is a spanning tree of H.

To prove the lemma, it is sufficient to show that the above construction of S_λ is equivalent to applying Krusal's minimum spanning tree algorithm on H under $\pi_\lambda(\cdot)$. To show this, it is sufficient to show that for each edge e of S_λ, when e is to be added to S_λ, $\pi_\lambda(e) \leq \pi_\lambda(x, y)$ is held for every edge (x, y), with $x \in J \cup \{r\}$ and $y \in J$ not being connected by any path of edges that have been added to S_λ. First, we know

that at this moment an edge denoted by e' is also to be added to $\hat{T}_\lambda$. By the following two cases, we know $\ell_\lambda(e') \leq \pi_\lambda(x, y)$:

- If $x \in J$, then x and y are not connected by any path of edges that have been added to $\hat{T}_\lambda$, and since $y \in J$ implies $\ell_\lambda(x, y) = \pi_\lambda(x, y)$, we have $\ell_\lambda(e') \leq \ell_\lambda(x, y) = \pi_\lambda(x, y)$.

- Otherwise, $x = r$, and thus there exists $i \in W$, such that $\ell(i, y) = \min\{\ell(i', y) : i' \in W\}$. This implies that $\pi_\lambda(r, y) = \ell(i, y) + \lambda$. Thus, we have that $\ell_\lambda(i, y) = \ell(i, y) \leq \pi_\lambda(r, y)$, and that $\ell_\lambda(r, i) = s(i) + \lambda \leq \pi_\lambda(r, y)$. Since r and y are not connected by any path of the selected edges in S_λ, we know either that i and y are not connected by any path of the selected edges in $\hat{T}_\lambda$, implying $\ell_\lambda(e') \leq \ell_\lambda(i, y) \leq \pi_\lambda(r, y)$, or that i and y are connected by a path of the selected edges in $\hat{T}_\lambda$ but r and i are not, implying $\ell_\lambda(e') \leq \ell_\lambda(r, i) \leq \pi_\lambda(r, y)$. Hence, we obtain that $\ell_\lambda(e') \leq \pi_\lambda(r, y) = \pi_\lambda(x, y)$.

Moreover, for the edge e, which is now to be added to S_λ, by the following two cases, we can see that $\pi_\lambda(e) \leq \ell_\lambda(e')$:

- If the endpoints of e are both in J, then $e = e'$, and $\pi_\lambda(e) = \ell(e)/2 = \ell(e')/2 = \ell_\lambda(e')$.

- Otherwise, e and e' can be represented by $e = (r, i)$ and $e' = (r, v)$ with $i \in J$, $v \in W$ and $\ell(v, i) = s(v)$. Thus, since $\pi_\lambda(r, i) = \min\{\ell(u', i) : u' \in W\} + \lambda$ and $\ell_\lambda(r, v) = s(v) + \lambda$, we have that $\pi_\lambda(e) = \pi_\lambda(r, i) \leq l(v, i) + \lambda = s(v) + \lambda = \ell_\lambda(r, v) = \ell_\lambda(e')$.

Hence, we obtain that $\pi_\lambda(e) \leq \ell_\lambda(e') \leq \pi_\lambda(x, y)$.

Therefore, S_λ can be constructed by applying Krusal's minimum spanning tree algorithm on H under $\pi_\lambda(\cdot)$, and so S_λ is a minimum spanning tree of H under $\pi_\lambda(\cdot)$. □

Lemma 3.3 implies that for any given $\lambda \geq 0$, the primal-dual schema used in the 2-approximation algorithm of [16] is equivalent to computing a minimum spanning tree of H under the revised edge weights $\pi_\lambda(\cdot)$. From this, we can prove as follows that the 2-approximation algorithm of [16] is equivalent to computing a minimum k-RDST of H under $\pi_0(\cdot)$ (for $\lambda = 0$).

Theorem 3.2. *Consider the value of λ_k determined in the algorithm of [16]. Then, S_{λ_k} is a minimum k-RDST of H under $\pi_0(\cdot)$.*

Proof. If T_0 consists of no more than k trees, then $\lambda_k = 0$, and the degree of the root r in S_0 does not exceed k. Since by Lemma 3.3, S_0 is a minimum spanning tree of H under $\pi_0(\cdot)$, we know that S_{λ_k} is a minimum k-RDST of H under $\pi_0(\cdot)$. Otherwise, for the value of λ_k determined in the algorithm of [16], T_{λ_k} must contain exactly k trees, implying that the degree of root r in S_{λ_k} equals k, and thus S_{λ_k} is a k-RDST of H. Consider any minimum k-RDST S^* of H under $\pi_0(\cdot)$. Since S^* is a spanning tree of H, by Lemma 3.3 we have that

$$\sum_{e \in E(S_{\lambda_k})} \pi_0(e) = \sum_{e \in E(S_{\lambda_k})} \pi_{\lambda_k}(e) - \lambda_k k \leq \sum_{e \in E(S^*)} \pi_{\lambda_k}(e) - \lambda_k k \leq \sum_{e \in E(S^*)} \pi_0(e).$$

Hence, S_{λ_k} is a minimum k-RDST of H under $\pi_0(\cdot)$. □

In light of Theorem 3.2, we can simplify the existing 2-approximation algorithm of [16] to a transformation from a minimum k-RDST of H, so that it consists of the following three steps:

Step 1. Construct H and $\pi_0(\cdot)$;

Step 2. Compute a minimum k-RDST S^* of H under $\pi_0(\cdot)$;

Step 3. Transform S^* to a Steiner forest $\hat{F}$ of G by replacing each edge (r, v) in $\hat{F}$ with (u, v), where $u \in W$ and $\ell(u, v) = \min\{(u', v) : u' \in W\}$, so that we have $\ell(\hat{F}) = \ell(S^*)$.

Since Lemma 3.1 indicates that a minimum k-RDST of H can be transformed from a minimum spanning tree of H, we obtain that the algorithm of [16] is also equivalent to a transformation from the minimum spanning tree of H under $\pi_0(\cdot)$. Moreover, based on this equivalence, we can provide a simpler proof of the approximation ratio 2 for the algorithm of [16], as demonstrated below.

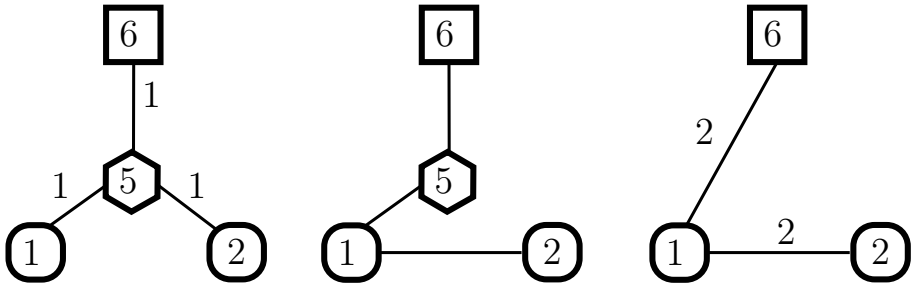
Consider the above 3-step algorithm. Let $\hat{F}$ indicate the k-median Steiner forest of G that it constructs, with $\ell(\hat{F}) = \ell(S^*)$, where S^* is a minimum k-DRST of H under edge weights $\pi_0(\cdot)$, and H, as defined in Section 3.2.1, is a complete graph on $J \cup \{r\}$. Consider an optimal k-median Steiner forest OPT of G. We thus establish Theorem 3.3 below, providing a new proof of the approximation ratio 2 for the algorithm of [16].

Theorem 3.3. $\ell(\hat{F}) \leq 2\ell(\text{OPT})$.

Proof. For each tree T of OPT, let w_T indicate its root, $V(T)$ the set of its vertices, $J(T)$ the set of its clients, $E(T)$ the set of its edges, and $E_J(T)$ the set of its edges (u, v) that join pairs of clients with $u \in J$ and $v \in J$. By the definition of π_λ for $\lambda = 0$,we have $\pi_0(E_J(T)) + \ell(E(T) \setminus E_J(T)) \leq \ell(E_J(T)) + \ell(E(T) \setminus E_J(T)) = \ell(T)$.

We first follow the procedure below to construct from T a new tree T' with $\pi_0(E_J(T')) + \ell(E(T') \setminus E_J(T')) \leq \pi_0(E_J(T)) + \ell(E(T) \setminus E_J(T))$, such that (i) $V(T')$ consists of only vertices in $J(T') \cup \{w_T\}$, and that (ii) T' contains at most one edge that w_T is incident with.

Start with $T' = T$. First, until T' satisfies the above condition (i), repeat the following to revise T' with $\pi_0(E_J(T'))+\ell(E(T')\setminus E_J(T'))$ always being non-increasing. Since condition (i) is not satisfied, it can be seen that there must exist a vertex $u \in V(T') \setminus J(T') \setminus \{w_T\}$ such that every child of u (if it exists) is a client in $J(T')$. Let $p(u)$ indicate the parent of u in T'. We revise T' by the following three cases: For case 1, where u has no child, we remove u and $(p(u), u)$ from T', and it is then easy to see that $\pi_0(E_J(T')) + \ell(E(T') \setminus E_J(T'))$ is not increasing. For case 2, where u has



(a) Initially, $T' = T$, where $w_T = 6$

(b) After replacing $(5, 2)$ with $(1, 2)$, as $u = 5$ has two children $v = 1$ and $v' = 2$.

(c) After removing 5 and replacing $(6, 5)$ and $(5, 1)$ with $(6, 1)$, as $u = 5$ has only one child $v = 1$, and $p(u) = 6$.

Figure 3.4: Illustration of the construction of T' from T in the proof of Theorem 3.3, where T is one of the trees of the optimal solution for the instance shown in Figure 3.1.

only one child denoted by v, we remove u and replace edges $(p(u), u)$ and (u, v) with $(p(u), v)$ in T', and by triangle inequality, we have that $\ell(p(u), v) \leq \ell(p(u), u) + \ell(u, v)$, which, together with $\pi_0(p(u), v) \leq \ell(p(u), v)$ if $p(u) \in J$ and $v \in J$, implies that $\pi_0(E_J(T')) + \ell(E(T') \setminus E_J(T'))$ is not increasing. For case 3, where u has at least two children, we let v denote the child of u with $\ell(u, v)$ minimized, and for each child v' of u other than v, replace (u, v') with (v, v'), and by triangle inequality, we have that $\pi_0(v, v') = \ell(v, v')/2 \leq [\ell(u, v) + \ell(u, v')]/2 \leq \ell(u, v')$, which implies that $\pi_0(E_J(T')) + \ell(E(T') \setminus E_J(T'))$ is not increasing. See Figure 3.4.

Next, until T' satisfies the above condition (ii), repeat the followings to revise T', with $\pi_0(E_J(T')) + \ell(E(T') \setminus E_J(T'))$ also always being non-increasing: Since condition (i) is satisfied but condition (ii) is not satisfied, we know that w_T must have at least two children that are all clients in $J(T')$. We can thus follow the step for the above case 3 to replace (w_T, v') with (v, v') for each child v' of u other than v, where v is the child of w_T with $\ell(w_T, v)$ minimized, and similarly, by triangle inequality, $\pi_0(E_J(T')) + \ell(E(T') \setminus E_J(T'))$ is also not increasing.

Hence, we obtain a tree T' from T satisfying both conditions (i) and (ii) with $\pi_0(E_J(T')) + \ell(E(T') \setminus E_J(T')) \leq \pi_0(E_J(T)) + \ell(E(T) \setminus E_J(T)) \leq \ell(T)$. Moreover, it

can be seen from the above construction that $J(T') = J(T)$. Thus, by condition (i), $V(T') = J(T) \cup \{w_T\}$.

Repeating the above procedure for every tree T in OPT, we can obtain a collection F' of at most k trees with $\pi_0(E_J(F')) + \ell(E(F') \setminus E_J(F')) \leq \ell(\text{OPT})$, such that F' covers all vertices of J, with each tree containing only vertices in $J \cup W$ and containing exactly one facility in W. Thus, by replacing facilities with r for all these trees, we can obtain a k-RDST S' of H with $\pi_0(S') \leq \pi_0(E_J(F')) + \ell(E(F') \setminus E_J(F')) \leq \ell(\text{OPT})$. Since S^* is a minimum k-RDST of H under $\pi_0(\cdot)$, we have $\pi_0(S^*) \leq \pi_0(S')$. Hence, by $\ell(\hat{F}) = \ell(S^*) \leq 2\pi_0(S^*)$, we obtain $\ell(\hat{F}) \leq 2\pi_0(S^*) \leq 2\pi_0(S') \leq 2\ell(\text{OPT})$. □

3.2.3.3 Improvements made by the new algorithm

We can now compare the solution $\hat{F}$ returned by the existing 2-approximation algorithm of [16] with the solution F returned by our newly proposed approximation algorithm. The following theorem shows that our algorithm always produces solutions of equal or better quality than the existing algorithm.

Theorem 3.4. $\ell(F) \leq \ell(\hat{F})$.

Proof. As we have shown earlier in Section 3.2.3.2 and Section 3.2.1, $\hat{F}$ can be obtained from a minimum k-RDST S^* of H under $\pi_0(\cdot)$ with $\ell(S^*) = \ell(\hat{F})$, and F is constructed from a minimum k-RDST T_H of H under $\ell(\cdot)$ with $\ell(T_H) = \ell(F)$. Thus, since S^* is also a k-RDST of H under $\ell(\cdot)$, we have $\ell(F) = \ell(T_H) \leq \ell(S^*) = \ell(\hat{F})$. □

Moreover, the following example shows that $\ell(F)$ can be strictly better than $\ell(\hat{F})$, and that the improvement can be up to 50% in some cases.

Example 3.4. Consider $J = \{1, 2, ..., k\}$ and $W = \{k+1, k+2, ..., 2k\}$, where $\ell(k+1, 1) = 1$, $\ell(k+v, v) = 1 + \epsilon$ for each $v \in \{2, ..., k\}$ and for any $\epsilon > 0$, and $\ell(u, v) = 2$ for other edges (u, v). Since edges in $\{(r, v) : v \in J\}$ form a minimum k-RDST of H under $\ell(\cdot)$, it can be seen that our algorithm produces a solution F

that consists of edges $(k+v, v)$ for $v \in J$ with $\ell(F) = k + (k-1)\epsilon$. Since edges in $\{(r,1)\} \cup \{(1,v) : 2 \leq v \leq k\}$ form a minimum k-RDST of H under $\pi_0(\cdot)$, it can be seen that the algorithm of [16] produces a solution $\hat{F}$ that consists of edges $(k+1, 1)$ and $(1, v)$ for $2 \leq v \leq k$ with $\ell(\hat{F}) = 2(k-1) + 1 = 2k - 1$. It can be seen that $[\ell(\hat{F}) - \ell(F)]/\ell(\hat{F}) = 1/2 - [1 + 2(k-1)\epsilon]/(4k-2)$, which can be arbitrarily close to 50% when k grows to infinity and ϵ goes to zero.

To demonstrate the algorithm improvement, we conduct numerical experiments on randomly generated instances. Following the method proposed by [34], we generated 43 classes randomly, with each class containing 20 instances for a specific combination of $|V|$ and $|J|$, and we set $W = V \setminus J$. To obtain the optimal solution (best lower bound) and best integer solution as the benchmark for comparison, we model the problem as a mixed integer programming (see Appendix A for details) and solve it by commercial solver CPLEX 12.6. However, since the problem is $\mathcal{NP}$-hard, for large instances with $|V|$ greater than 30, it is difficult to obtain a better lower bound which is close to the optimal solution. Our preliminary results on these instances show that even after running CPLEX for two hours, the gap between best lower bound and best integer solution is still larger than 25%. Therefore, our experiments only use CPLEX to solve small instances, with a time limit of two hours, and compare the returned optimal solution (best lower bound) or best integer solution with the solutions F and $\hat{F}$, respectively. While for large instances, only the relative solution improvement made by F over $\hat{F}$ is evaluated.

For small instances with $|V|$ only in $\{10, 15, 20, 25\}$, we compare the solutions with the best lower bound returned by CPLEX. The optimality gap of the solutions is shown in Table 3.1, where columns 4-13 present the average optimality gap of the two approximation algorithms over the 20 instances for different values of ρ in $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ with $k = \lceil \rho |W| \rceil$ holds, respectively. The results show that the solution produced by our new algorithm (New) is close to the optimal solution,

Table 3.1: Computational results for the optimality gap (%) on randomly generated small instances.

$\|V\|$	$\|J\|$	$\|W\|$	$\rho = 0.1$		$\rho = 0.3$		$\rho = 0.5$		$\rho = 0.7$		$\rho = 0.9$	
			New	CS	New	CS	New	CS	New	CS	New	CS
10	5	5	3.6	3.6	3.9	11.8	3.0	17.9	0.0	15.4	0.0	16.0
15	10	5	2.4	2.4	2.6	6.3	2.4	10.6	0.5	11.3	0.6	11.0
15	5	10	4.6	4.6	3.8	12.8	0.0	17.8	0.0	16.6	0.0	19.2
20	10	10	2.5	2.5	4.3	10.2	1.2	16.2	0.2	16.2	0.0	14.4
20	5	15	5.9	9.7	0.0	13.2	0.0	16.4	0.0	20.2	0.0	19.4
25	15	10	17.7	17.7	7.8	12.1	2.7	12.8	0.9	15.4	0.3	14.3
25	10	15	3.3	3.8	2.4	16.1	0.0	19.7	0.0	20.3	0.0	20.7
Average			5.7	6.3	3.6	11.8	1.3	15.9	0.2	16.5	0.1	16.4

especially when ρ is large. However, the solution produced by the approximation algorithm of [16] (CS) is far away from the optimal solution. For example, when $\rho = 0.9$, our algorithm can return optimal solutions for most of instances while the average gap of the algorithm in [16] has 16.4%. Moreover, it can be seen that our new algorithm always outperforms the algorithm of [16], that is, for our algorithm the average optimality gap for all instance classes is less than 5.7% but it is greater than 6.3% for the algorithm of [16]. This result is consistent with the theoretical analysis in Theorem 3.4.

For small instances with $|V|$ in $\{30, 35, 40\}$, we compare the solutions with the best upper bound (UB) returned by CPLEX. The upper bound gap, which is measured by $[\ell(F) - \ell(UB)]/\ell(F) \cdot 100\%$ and $[\ell(\hat{F}) - \ell(UB)]/\ell(\hat{F}) \cdot 100\%$, is shown in Table 3.2, where columns 4-13 present the average upper bound gap of the two approximation algorithms. The results show that our new algorithm can effectively produce solutions for these instances in a short running time (less than 0.1 seconds) with quality as good as those of CPLEX (two hours). In particular, the average gap is less than 1.2%, and can even be negative, which implies that our solutions are better than those returned by CPLEX. However, the solutions $\hat{F}$ are always worse than those of CPLEX for all

Table 3.2: Computational results for the upper bound gap (%) on randomly generated small instances.

$\|V\|$	$\|J\|$	$\|W\|$	$\rho = 0.1$		$\rho = 0.3$		$\rho = 0.5$		$\rho = 0.7$		$\rho = 0.9$	
			New	CS	New	CS	New	CS	New	CS	New	CS
30	20	10	1.6	1.6	0.3	4.6	1.5	10.8	0.7	12.1	0.8	13.1
30	15	15	0.4	0.4	3.0	9.4	1.6	16.3	0.0	17.6	0.0	14.7
35	25	10	-0.3	-0.3	0.2	3.9	0.5	7.0	-2.0	7.8	0.0	10.4
35	20	15	0.0	0.0	1.6	8.5	1.1	12.8	0.0	15.3	-2.8	13.3
35	15	20	0.8	0.8	3.5	17.5	0.3	18.7	0.0	19.8	0.0	21.4
40	25	15	-4.5	-4.5	-2.8	1.5	-2.2	6.6	-2.1	10.1	-2.5	9.1
40	20	20	-3.0	-2.3	2.4	11.7	0.5	14.7	-0.1	16.5	0.0	15.0
40	15	25	0.5	1.2	1.6	16.1	0.0	19.9	0.0	19.7	0.0	19.2
Average			-0.6	-0.4	1.2	9.1	0.4	13.4	-0.4	14.9	-0.6	14.5

the instances when $\rho \geq 0.3$.

For large instances, we only evaluate the average improvement made by F over $\hat{F}$, which is measured by the gap ratio $[\ell(\hat{F}) - \ell(F)]/\ell(\hat{F}) \cdot 100\%$. The computational results are shown in Table 3.3, where columns 5–11 present the average gap ratios over the 20 instances of each class for different values of ρ ranging from 0.2 to 0.8, respectively. From the results, it can be seen that (i) the solution F produced by our new algorithm is always better than the solution $\hat{F}$ produced by the algorithm of [16], and that (ii) the average improvement increases from 4.3% to 11.4% as ρ increases from 0.2 to 0.8. This is mainly because the improvement $[\ell(\hat{F}) - \ell(F)]$ equals $[\ell(S^*) - \ell(T_H)]$, and both S^* and T_H are minimum k-RDSTs of H but under different edge weights, i.e., $\pi_0(\cdot)$ and $\ell(\cdot)$, respectively. Since $\pi_0(r, v)$ equals $\ell(r, v)$ for $v \in J$, and $\pi_0(u, v)$ is only half of $\ell(u, v)$ for $u \in J$ and $v \in J$, those root edges (r, v) in T_H may not be preferred by S^*. A further investigation on the numerical results reveals that S^* contains significantly fewer root edges than T_H, and this difference becomes greater as k increases. This results in the increase of $[\ell(S^*) - \ell(T_H)]$, which results in the increase of the improvement $[\ell(\hat{F}) - \ell(F)]$.

Table 3.3: Computational results for the gap ratio $[\ell(\hat{F}) - \ell(F)]/\ell(\hat{F}) \cdot 100\%(\%)$ on randomly generated large instances.

Class	$\|V\|$	$\|J\|$	$\|W\|$	ρ						
				0.2	0.3	0.4	0.5	0.6	0.7	0.8
1	50	40	10	0.3	1.4	2.3	3.5	4.2	5.4	6.6
2	60	50	10	0.1	1.5	1.5	3.1	3.5	4.3	4.3
3	60	30	30	3.7	9.0	11.4	12.4	13.0	13.1	13.7
4	70	40	30	2.0	5.9	8.8	9.8	11.3	11.2	10.6
5	80	50	30	2.4	5.4	8.2	8.5	8.7	9.3	8.5
6	100	50	50	5.4	10.7	12.5	12.2	11.9	12.6	14.0
7	70	60	10	0.2	1.1	1.6	2.2	2.9	3.1	3.6
8	90	60	30	2.7	4.9	7.6	7.9	8.3	8.7	8.4
9	110	60	50	5.4	9.8	13.0	13.2	13.2	13.1	12.2
10	80	70	10	0.4	1.0	1.6	2.3	2.6	3.0	3.4
11	100	70	30	2.4	5.2	7.3	8.3	8.3	8.8	8.8
12	120	70	50	5.3	9.1	12.7	13.0	13.8	13.4	13.9
13	140	70	70	8.6	14.4	17.0	16.6	16.7	16.2	17.9
14	90	80	10	8.6	14.4	17.0	16.6	16.7	16.2	17.9
15	110	80	30	2.8	4.9	7.1	8.2	8.4	8.4	9.0
16	130	80	50	5.2	9.1	12.5	13.3	13.9	13.6	13.1
17	150	80	70	8.2	13.6	17.2	17.9	17.8	17.2	18.2
18	100	90	10	0.5	1.0	1.6	2.2	2.5	2.6	3.1
19	120	90	30	2.9	4.8	6.3	7.9	8.3	8.4	8.3
20	140	90	50	5.2	8.5	11.4	13.1	13.0	12.8	13.0
21	160	90	70	7.9	12.4	16.7	17.7	17.8	16.8	17.7
22	180	90	90	10.2	16.4	21.3	21.3	21.0	20.9	20.3
23	110	100	10	0.5	1.0	1.6	2.1	2.2	3.0	2.4
24	130	100	30	2.6	4.4	6.2	7.9	8.2	7.7	8.5
25	150	100	50	4.9	8.0	11.0	12.2	13.1	13.4	13.1
26	170	100	70	7.4	11.4	15.6	17.4	17.2	17.0	17.3
27	190	100	90	9.7	15.1	19.8	20.3	20.5	20.6	20.3
	Average			4.3	7.6	10.0	10.8	11.1	11.1	11.4

3.3 Polynomial Time Algorithms for Special Cases

In this section, we present new polynomial time algorithms that can solve two non-trivial special cases of the k-median Steiner forest problem to optimality.

3.3.1 When $J = V$

First, we show that solving the k-median Steiner forest problem with $J = V$ is equivalent to finding a minimum weighted basis for a matroid. Consider a collection $\mathcal{S}$ of edge subsets of E, such that for each $S \in \mathcal{S}$ the subgraph (V, S) satisfies that it has no cycles, and that at least k of its connected components each contain at least one facility in W.

The system $(E, \mathcal{S})$ is a matroid, due to the following arguments. It is easy to see that $\emptyset \in \mathcal{S}$, and that if $X \subseteq Y \in \mathcal{S}$, then $X \in \mathcal{S}$. Thus, to prove that $(E, \mathcal{S})$ is a matroid, we only need to show that, for each subset $Y \subseteq E$, every maximal independent subset X of Y, which is also called a *basis* of Y, has the same cardinality. For each $S \subseteq E$, let α_S and β_S denote the numbers of connected components of the subgraph (V, S) that contain no facility, and that contain at least one facility, respectively. Since X contains no cycle, we have $|X| = |V| - \alpha_X - \beta_X$. Since X is a maximal independent subset of Y, we have that $\alpha_X = \alpha_Y$ and $\beta_X = \max\{k, \beta_Y\}$. Hence, $|X|$ is always equal to $(|V| - \alpha_Y - \max\{k, \beta_Y\})$, and so $(E, \mathcal{S})$ is a matroid.

Since G is a complete graph, implying that $\alpha_E = 0$ and $\beta_E = 1$, every basis of $(E, \mathcal{S})$ consists of $|V| - k$ edges that correspond to a k-median Steiner forest of G. Moreover, it is easy to see that for every k-median Steiner forest of G, its edge set is a basis of $(E, \mathcal{S})$. Hence, we obtain the following theorem, which implies that the k-median Steiner forest problem with $J = V$ can be solved to optimality by finding a minimum weighted basis of the matroid $(E, \mathcal{S})$, which can be obtained in $O(n^2 \log n)$ time by a greedy algorithm known in the literature [50].

Theorem 3.5. *Solving the k-median Steiner forest problem with $J = V$ is equivalent to finding a minimum weighted basis of $(E, \mathcal{S})$.*

Next, consider a complete graph H extended from G, where $V(H) = V \cup \{r\}$, $E(H) = E \cup \{(r, u) : u \in W\}$, and $\ell(r, u) = 0$ for $u \in W$. We can show as follows that, when $J = V$, an optimal k-median Steiner forest F of G can be transformed from a minimum k-RDST T of H. First, by eliminating vertex r and all edges (r, u) with $u \in W$ from T, we can obtain a tree collection F that covers all vertices in V. Since the degree of r in T does not exceed k, we know that F has at most k trees, with each tree containing a vertex in W as the root. Hence, F is a k-median Steiner forest of G, and it can be seen that $\ell(F) = \ell(T)$. Since OPT denotes an optimal k-median Steiner forest of G, we have that $\ell(\text{OPT}) \leq \ell(F) = \ell(T)$. Moreover, from OPT we can obtain a k-RDST of H by joining the root of each tree in OPT to r. This implies that $\ell(T) \leq \ell(\text{OPT})$. Hence, we have $\ell(F) = \ell(T) = \ell(\text{OPT})$, and so F is an optimal k-median Steiner forest of G. Since T can be obtained by an $O(n^2)$-time transformation from a minimum spanning tree T^* of H (by Lemma 3.1), F can also be obtained by an $O(n^2)$-time transformation from T^*. Since T^* can be obtained in $O(\alpha(n^2, n)n^2)$ time [21], F can also be obtained in $O(\alpha(n^2, n)n^2)$ time, faster than by directly applying the greedy algorithm of finding a minimum weighted basis for a matroid.

Example 3.5. Consider the instance in Figure 3.1, but let $J = V = \{1, 2, ..., 8\}$. It can be seen that H contains vertices $1, 2, ..., 8$ as well as a new vertex r, and that edges in $\{(r, 6), (r, 7), (6, 5), (5, 1), (5, 2), (7, 3), (7, 4), (4, 8)\}$ form a minimum k-RDST of H. By removing $(r, 6)$ and $(r, 7)$ we obtain an optimal k-median Steiner forest of G of total edge weight equal to 7. This can also be obtained by applying the greedy algorithm of finding a minimum weighted basis for a matroid, which selects edges $(1, 5), (2, 5), (5, 6), (3, 7), (4, 8), (4, 7)$ sequentially to form the optimal k-median Steiner forest.

3.3.2 When vertices are located in a tree-shaped network

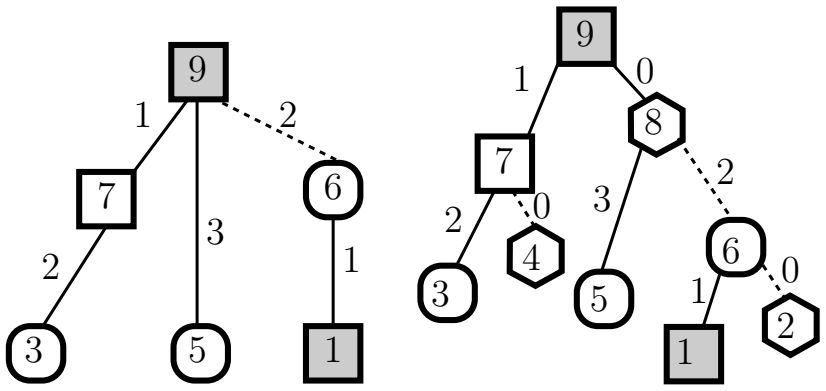
Suppose that vertices of G are all located on a tree T with a root $r \in W$, so that the weight of each edge of G equals the total edge weight of the simple path that connects the endpoints in T. For this special case, the k-median Steiner forest problem on G is equivalent to that on T, and we can solve it to optimality by the $O(kn)$-time algorithm below, which improves on the existing fastest algorithm of [76] that runs in $O(n2^{6k})$ time.

For each edge (u, v) of T, we still use $\ell(u, v)$ to indicate its edge weight. First, without loss of generality, assume that J and W are disjoint. Moreover, the same as only considering the standard instances in Chapter 2 (condition 3 of Definition 2.1), we assume that the tree T considered is a full binary tree, where each vertex other than the leaves has exactly two children, due to the following transformation, similar to the proof of Theorem 2.1 in Chapter 2 as well as the transformations in [71, 76]:

1. For any non-leaf vertex v with only one child, we can add a new vertex s as a child of v with $\ell(v, s) = 0$. Repeat this until every non-leaf v in G has at least two children.

2. For any non-leaf vertex v with more than two children, as denoted by $u_1, u_2, ..., u_q$ with $q \geq 3$, we can add a new vertex s as a second child of v with $\ell(v, s) = 0$, and add edges (s, u_j) with $\ell(s, u_j) = \ell(v, u_j)$ to replace (v, u_j) for $2 \leq j \leq q$, so that $u_2, ..., u_q$ become children of s. Repeat this until every non-leaf vertex v has exactly two children.

Notice that J and W are not changed. It can be seen that each k-median Steiner forest for the original tree corresponds to a k-median Steiner forest for the transformed tree with equal total edge weights, and vice versa. See Figure 3.5, where the instance on the right is on a full binary tree, and it is equivalently transformed from the instance on the left. Hence, we can assume without loss of generality that T is a full binary



(a) An instance on a general tree. (b) An instance on a full binary tree.

Figure 3.5: Two equivalent instances with $k = 2$ and $r = 9$ for the special case of the problem: Vertices are located on a tree with clients shown in cycles and facilities shown in squares; the optimal k-median Steiner forests for both instances are shown in solid lines with facilities 1 and 9 open, and with total edge weights both equal to 7.

tree.

Second, from any k-median Steiner forest F of T, we can obtain another k-median Steiner forest of T as follows, such that trees of the new forest are all vertex-disjoint, without increasing the total edge weight: If trees in F are all vertex-disjoint, then F is the forest we need. Otherwise, we can repeatedly combine those trees in F that share some vertices into one tree, until trees in F are all vertex-disjoint.

Third, in light of the above observation we know that to find an optimal k-median Steiner forest of T, it is equivalent to selecting a collection of at most k vertex-disjoint subtrees of T with the total edge weight minimized, such that it covers all the clients and contains at least one facility in each subtree. The latter problem can be solved to optimality by dynamic programming, as follows.

For each vertex v, let T_v indicate the subtree of T rooted at v that contains v and all descendants of v. For each integer $q \in \{0, 1, ..., k\}$, and $\vec{a} = \langle a_2 a_1 a_0 \rangle$ with $a_i \in \{0, 1\}$ for $0 \leq i \leq 2$, we use $(v, q, \vec{a})$ to denote a subproblem that aims to minimize the total edge weight of a collection of q selected vertex-disjoint subtrees of T_v such that: (i) Each client of T_v belongs to one and only one selected subtree;

(ii) each selected subtree that does not include v contains at least one facility; (iii) v belongs to a selected subtree if, and only if, $a_2 = 1$; (iv) v belongs to a selected subtree that contains at least one client if, and only if, $a_1 = 1$; and (v) v belongs to a selected subtree that contains at least one facility if, and only if, $a_0 = 1$. It can be seen that if $a_2 = 0$, then unless $a_1 = a_0 = 0$, no feasible solutions to the subproblem $(v, q, \vec{a})$ exist. Hence, it is sufficient to consider only those subproblems with $\vec{a} \in A$, where $A = \{\langle 000\rangle, \langle 100\rangle, \langle 110\rangle, \langle 101\rangle, \langle 111\rangle\}$.

For example, consider the instance in Figure 3.5(b), for which a subtree formed by edges in $\{(8, 5), (8, 6)\}$ and vertices in $\{5, 6, 8\}$ is feasible to subproblem $(8, 1, \langle 110\rangle)$, and for which a collection of two vertex-disjoint subtrees formed by edges in $\{(6, 1), (6, 2)\}$ and vertices in $\{1, 2, 5, 6\}$ is not feasible to subproblem $(8, 2, \langle 000\rangle)$, because the subtree $(\{5\}, \emptyset)$ does not include 8 or any facility.

Let $S(v, q, \vec{a})$ indicate the optimal value of the subproblem $(v, q, \vec{a})$. Since $r \in W$ and $r \notin J$, the minimum value of $S(r, q, \vec{a})$ over $1 \leq q \leq k$ and $\vec{a} \in \{\langle 000\rangle, \langle 111\rangle\}$ equals the total edge weight of the minimum k-median Steiner tree of T.

We next present a dynamic programming algorithm to compute $S(v, q, \vec{a})$ recursively for $v \in V$, from leaves to the root r. For each leaf v of T, we can compute $S(v, q, \vec{a})$ with $0 \leq q \leq k$ and $\vec{a} \in A$ by the following three cases:

- Case 1: $v \in J$ is a client. Since only the subtree $(\{v\}, \emptyset)$ can cover v, and since $v \in J$, we have that $S(v, q, \vec{a}) = 0$ if $q = 1$ and $\vec{a} = \langle 110\rangle$, and that $S(v, q, \vec{a}) = \infty$ otherwise.

- Case 2: $v \in W$ is a facility. Since only the subtree $(\{v\}, \emptyset)$ can cover v, and since $v \in W$, we have that $S(v, q, \vec{a}) = 0$ if $q = 1$ and $\vec{a} = \langle 101\rangle$, that $S(v, q, \vec{a}) = 0$ if $q = 0$ and $\vec{a} = \langle 000\rangle$, and that $S(v, q, \vec{a}) = \infty$ otherwise.

- Case 3: $v \in V \setminus J \setminus W$. Since v is neither a client nor a facility, we have that $S(v, q, \vec{a}) = 0$ if $q = 0$ and $\vec{a} = \langle 000\rangle$, that $S(v, q, \vec{a}) = 0$ if $q = 1$ and $\vec{a} = \langle 100\rangle$,

Table 3.4: Values of $S(v, q, \vec{a})$ for $\vec{a} \in A$, $q \in \{0, 1\}$, and each leaf $v \in \{1, 2, 3, 4, 5\}$ of the tree of the instance shown in Figure 3.5(b), with $k = 2$.

v	$\vec{a}$:	$q = 0$					$\vec{a}$:	$q = 1$				
		$\langle 000 \rangle$	$\langle 100 \rangle$	$\langle 110 \rangle$	$\langle 101 \rangle$	$\langle 111 \rangle$		$\langle 000 \rangle$	$\langle 100 \rangle$	$\langle 110 \rangle$	$\langle 101 \rangle$	$\langle 111 \rangle$
3,5		∞	∞	∞	∞	∞		∞	∞	0	∞	∞
1		0	∞	∞	∞	∞		∞	∞	∞	0	∞
2,4		0	∞	∞	∞	∞		∞	0	∞	∞	∞

and that $S(v, q, \vec{a}) = \infty$ otherwise.

For example, consider the instance in Figure 3.5(b) with $k = 2$, for which Table 3.4 presents the values of $S(v, q, \vec{a})$ for each leaf $v \in \{1, 2, 3, 4, 5\}$ of the tree, where $q \in \{0, 1\}$ and $\vec{a} \in A$, and we know $S(v, 2, \vec{a}) = \infty$ for each leaf $v \in \{1, 2, 3, 4, 5\}$ and $\vec{a} \in A$.

For each non-leaf v of T, let u_1 and u_2 denote its left and right children. Consider each subproblem $(v, q, \vec{a})$ with $q = 0, 1, ..., k$ and $\vec{a} \in A$. If $\vec{a} = \langle 000 \rangle$, implying that v does not belong to any selected subtree of T_v, then an optimal solution to $(v, q, \vec{a})$ exists only when v is not a client, and if such an optimal solution exists, then we can partition it into two collections of selected subtrees that are optimal to subproblems $(u_1, q_1, \vec{a}_1)$ and $(u_2, q_2, \vec{a}_2)$, respectively, for certain q_1 and q_2 with $q_1 + q_2 = q$, and for certain $\vec{a}_1$ and $\vec{a}_2$ in A. Moreover, for $j \in \{1, 2\}$, since (v, u_j) is not selected, the selected subtree that contains u_j (if any) must contain at least one client and at least one facility. In other words, for any optimal solution to the subproblem $(u_j, q_j, \vec{a}_j)$, it can be part of an optimal solution to $(v, q, \vec{a})$ only if $\vec{a}_j \in \{\langle 000 \rangle, \langle 111 \rangle\}$. Thus, if $v \in J$, we obtain that $S(v, q, \langle 000 \rangle) = \infty$, and otherwise,

$$\begin{aligned} S(v, q, \langle 000 \rangle) \quad &= \quad \min S(u_1, q_1, \vec{a}_1) + S(u_2, q_2, \vec{a}_2) \\ \text{s.t.} \quad & \quad q_1 + q_2 = q, \\ & \quad 0 \leq q_1, q_2 \leq q, \\ & \quad \vec{a_1} \in \{\langle 000 \rangle, \langle 111 \rangle\}, \vec{a_2} \in \{\langle 000 \rangle, \langle 111 \rangle\}. \end{aligned}$$

For example, consider the instance in Figure 3.5(b), in which it can be seen that $S(6, 1, \langle 000 \rangle) = \infty$ since 6 is a client, and that by Table 3.4, $S(7, 1, \langle 000 \rangle) = \infty$ since $S(v, 1, \langle 000 \rangle) = S(v, 1, \langle 111 \rangle) = \infty$ for both $v \in \{3, 4\}$.

Next, consider the situation where $\vec{a} \in A \setminus \{\langle 000 \rangle\} = \{\langle 100 \rangle, \langle 110 \rangle, \langle 101 \rangle, \langle 111 \rangle\}$, implying that $a_2 = 1$, and thus v belongs to a selected subtree of T_v. Depending on whether or not edge (v, u_1) and/or edge (v, u_2) are selected, there are four possibilities for v to be connected to a selected subtree of T_v. We can thus decompose the subproblem $(v, q, \vec{a})$ into the following four restricted subproblems denoted by $(v, q, \vec{a})_i$ with $1 \leq i \leq 4$, so that representing the optimal value of each restricted subproblem by $S_i(v, q, \vec{a})$, we have

$$S(v, q, \vec{a}) \quad = \quad \min_{1 \leq i \leq 4} S_i(v, q, \vec{a}). \tag{3.2}$$

For the restricted subproblem $(v, q, \vec{a})_1$, it is restricted to a constraint where neither edge (v, u_1) nor (v, u_2) is selected. Thus, the subtree $(\{v\}, \emptyset)$ must be selected. Define

$$\vec{b} := \langle 1 b_1 b_0 \rangle, \tag{3.3}$$

where $b_1 = 1$ if v is a client, and $b_1 = 0$ otherwise, and $b_0 = 1$ if v is a facility, and $b_0 = 1$ otherwise. If $\vec{a} \neq \vec{b}$, then no feasible solution exists to the restricted subproblem $(v, q, \vec{a})_1$, implying that $S_1(v, q, \vec{a}) = \infty$. Otherwise, $\vec{a} = \vec{b}$, and then, for any optimal solution to $(v, q, \vec{b})_1$, it can be partitioned into three components, including a vertex v, a collection of subtrees that is optimal to subproblem $(u_1, q_1, \vec{a}_1)$, and a collection of subtrees that is optimal to subproblem $(u_2, q_2, \vec{a}_2)$, for certain q_1 and q_2 with $q_1 + q_2 + 1 = q$, and for certain $\vec{a}_1 \in A$ and $\vec{a}_2 \in A$. Moreover, for each $j \in \{1, 2\}$, since (v, u_j) is not selected, by following the same argument used earlier for $\vec{a}_j$ in solving $S(v, q, \langle 000 \rangle)$, we know that for any optimal solution to $(u_j, q_j, \vec{a}_j)$,

it can be part of an optimal solution to $(v, q, \vec{a})_1$ only if $\vec{a}_j \in \{\langle 000\rangle, \langle 111\rangle\}$. Hence, we obtain that

$$\begin{aligned} S_1(v, q, \vec{b}) &= \min S(u_1, q_1, \vec{a}_1) + S(u_2, q_2, \vec{a}_2) \\ \text{s.t.} \quad & q_1 + q_2 + 1 = q, \\ & 0 \leq q_1, q_2 \leq q, \\ & \vec{a_1} \in \{\langle 000\rangle, \langle 111\rangle\}, \vec{a_2} \in \{\langle 000\rangle, \langle 111\rangle\}, \end{aligned}$$

where $S_1(v, q, \vec{b}) = \infty$ if $q = 0$. For example, consider the instance in Figure 3.5(b) with $v = 6$, $u_1 = 1$ and $u_2 = 2$, for which $\vec{b} = \langle 110\rangle$, and it can be seen that $S_1(6, 1, \langle 111\rangle) = \infty$ since $\vec{b} \neq \langle 111\rangle$, and that $S_1(6, 1, \langle 110\rangle) = 0$, since by Table 3.4 $S(1, 0, \langle 000\rangle) = S(2, 0, \langle 000\rangle) = 0$.

For the restricted subproblem $(v, q, \vec{a})_2$, it is restricted to a constraint where (v, u_1) is selected but (v, u_2) is not. Thus, for any optimal solution to $(v, q, \vec{a})_1$, it can be partitioned into three components, including (v, u_1), a collection of subtrees that is optimal to subproblem $(u_1, q_1, \vec{a}_1)$, and a collection of subtrees that is optimal to subproblem $(u_2, q_2, \vec{a}_2)$, for certain q_1 and q_2 with $q_1 + q_2 = q$, and for certain $\vec{a}_1 \in A$ and $\vec{a}_2 \in A$ with $\vec{a}_1 \vee \vec{b} = \vec{a}$, where b is as defined in (3.3). Since (v, u_2) is not selected, by following the same argument used earlier for $\vec{a}_2$ in solving $S(v, q, \langle 000\rangle)$, we know that for any optimal solution to $(u_2, q_2, \vec{a}_2)$, it can be part of an optimal solution to $(v, q, \vec{a})_2$ only if $\vec{a}_2 \in \{\langle 000\rangle, \langle 111\rangle\}$. Moreover, since (v, u_1) is selected, we know that for any optimal solution to $(u_1, q_1, \vec{a}_1)$, it can be part of an optimal solution to $(v, q, \vec{a})_2$ only if $q_1 \geq 1$ and $a_{1,2} = 1$, or in other words, $q_1 \geq 1$ and $\vec{a}_1 \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\}$. Hence, we obtain that

$$
\begin{aligned}
S_2(v, q, \vec{a}) \quad &= \quad \min \ell(v, u_1) + S(u_1, q_1, \vec{a}_1) + S(u_2, q_2, \vec{a}_2) \\
\text{s.t.} \quad & \quad q_1 + q_2 = q,\ \vec{a}_1 \vee \vec{b} = \vec{a}, \\
& \quad 1 \leq q_1 \leq q,\ 0 \leq q_2 \leq q, \\
& \quad \vec{a_1} \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\},\ \vec{a_2} \in \{\langle 000\rangle, \langle 111\rangle\},
\end{aligned}
$$

where $S_2(v, q, \vec{a}) = \infty$ if $q = 0$ or no $\vec{a}_1 \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\}$ satisfies that $\vec{a}_1 \vee \vec{b} = \vec{a}$. For example, consider the instance in Figure 3.5(b) with $v = 6$, $u_1 = 1$ and $u_2 = 2$, for which $\vec{b} = \langle 110\rangle$, and it can be seen that $S_2(6, 1, \langle 111\rangle) = \ell(6, 1) = 1$, since by Table 3.4 $S(1, 1, \langle 101\rangle) = S(2, 0, \langle 000\rangle) = 0$.

For the restricted subproblem $(v, q, \vec{a})_3$, it is restricted to a constraint where (v, u_2) is selected but (v, u_1) is not. Since the subproblem $(v, q, \vec{a})_3$ can be transformed to $(v, q, \vec{a})_2$ by switching the notation u_1 and u_2, similarly to $S_2(v, q, \vec{a})$, we obtain that

$$
\begin{aligned}
S_3(v, q, \vec{a}) \quad &= \quad \min \ell(v, u_2) + S(u_1, q_1, \vec{a}_1) + S(u_2, q_2, \vec{a}_2) \\
\text{s.t.} \quad & \quad q_1 + q_2 = q,\ \vec{a}_2 \vee \vec{b} = \vec{a}, \\
& \quad 0 \leq q_1 \leq q,\ 1 \leq q_2 \leq q, \\
& \quad \vec{a_1} \in \{\langle 000\rangle, \langle 111\rangle\},\ \vec{a_2} \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\},
\end{aligned}
$$

where $S_3(v, q, \vec{a}) = \infty$ if $q = 0$, or no $\vec{a}_2 \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\}$ satisfies that $\vec{a}_2 \vee \vec{b} = \vec{a}$. For example, consider the instance in Figure 3.5(b) with $v = 6$, $u_1 = 1$ and $u_2 = 2$, for which $\vec{b} = \langle 110\rangle$, and it can be seen that $S_3(6, 1, \langle 111\rangle) = \infty$, since by Table 3.4 $S(1, 0, \langle 111\rangle) = S(2, 1, \langle 101\rangle) = S(2, 1, \langle 111\rangle) = \infty$.

Finally, for the restricted subproblem $(v, q, \vec{a})_4$, it is restricted to a constraint where both (v, u_1) and (v, u_2) are selected. Thus, for any optimal solution to $(v, q, \vec{a})_1$, it can be partitioned into three components, including edges (v, u_1) and (v, u_2), a collection of subtrees that is optimal to subproblem $(u_1, q_1, \vec{a}_1)$, and a collection of subtrees that is optimal to subproblem $(u_2, q_2, \vec{a}_2)$, for certain q_1 and q_2 with $q_1 + q_2 - 1 = q$,

and for certain $\vec{a}_1 \in A$ and $\vec{a}_2 \in A$ with $\vec{a}_1 \vee \vec{a}_2 \vee \vec{b} = \vec{a}$. Moreover, for each $j \in \{1, 2\}$, since (v, u_j) is selected, by following the same argument used earlier for $\vec{a}_1$ in computing $S_1(v, q, \vec{b})$, we know that for any optimal solution to the subproblem $(u_j, q_j, \vec{a}_j)$, it can be part of an optimal solution to $(v, q, \vec{a})_4$ only if $q_j \geq 1$ and $\vec{a}_j \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\}$. Hence, we obtain that

$$\begin{aligned} S_4(v, q, \vec{a}) &= \min \ell(v, u_1) + \ell(v, u_2) + S(u_1, q_1, \vec{a}_1) + S(u_2, q_2, \vec{a}_2) \\ \text{s.t.} \quad & q_1 + q_2 - 1 = q, \ \vec{a}_1 \vee \vec{a}_2 \vee \vec{b} = \vec{a}, \\ & 1 \leq q_j \leq q, \\ & \vec{a_j} \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\}, \text{ for } j \in \{1, 2\}, \end{aligned}$$

where $S_4(v, q, \vec{a}) = \infty$ if $q = 0$, or no $\vec{a_j} \in \{\langle 100\rangle, \langle 101\rangle, \langle 110\rangle, \langle 111\rangle\}$ for $j = 1$ and $j = 2$ satisfy that $\vec{a}_1 \vee \vec{a}_2 \vee \vec{b} = \vec{a}$. For example, consider the instance in Figure 3.5(b) with $v = 6$, $u_1 = 1$ and $u_2 = 2$, for which $\vec{b} = \langle 110\rangle$, and it can be seen that $S_4(6, 1, \langle 111\rangle) = \ell(6, 1) + \ell(6, 2) = 1$, since by Table 3.4 $S(1, 1, \langle 101\rangle) = S(2, 1, \langle 100\rangle) = 0$. Hence, from (3.2), we obtain that $S(6, 1, \langle 111\rangle) = \min\{\infty, 1, \infty, 1\} = 1$.

We can now follow the above dynamic programming algorithm to solve subproblems $(v, q, \vec{a})$ recursively for all vertices $v \in V$, from leaves to the root r, for each $q = 0, 1, ..., k$, and for each $\vec{a} \in A$. As shown earlier, by taking the minimum value of $S(r, q, \vec{a})$ over $1 \leq q \leq k$ and $\vec{a} \in \{\langle 000\rangle, \langle 111\rangle\}$, we can obtain the total edge weight of the optimal k-median Steiner forest of T. For example, consider the instance in Figure 3.5(b) with $r = 9$ and $k = 2$, for which we can follow this dynamic programming algorithm to compute subproblems for vertices $1, 2, 3, ..., r = 9$, sequentially, and to obtain that $S(r, 2, \langle 111\rangle) = 7$ is the total edge weight of the optimal k-median Steiner forest.

Moreover, the dynamic programming algorithm runs in $O(kn)$ time, as shown below.

Theorem 3.6. *When vertices are located on a tree T, the k-median Steiner forest problem can be solved to optimality in $O(kn)$ time.*

Proof. We have shown that for this special case, an optimal k-median Steiner forest can be obtained by a dynamic programming algorithm, which solves at most $O(kn)$ subproblems $(v, q, \vec{a})$ recursively. Since $q \leq k$, by definition it can be seen that each subproblem $(v, q, \vec{a})$ can be solved in $O(k)$ time. Thus, the total time complexity of the algorithm is $O(k^2 n)$.

It can further be shown as follows that the total time complexity is $O(kn)$. Similar to the analysis in [71], we define that a vertex v is *rich* if it is not a leaf, and for each of its children u_j with $j \in \{1, 2\}$, the subtree T_{u_j} rooted at u_j contains at least $k/2$ vertices. By Lemma 1 in [71], we know that the number of rich vertices is bounded above by $2n/k$. This implies that the total time complexity for solving subproblems $(v, q, \vec{a})$ with v being a rich vertex is $O(kn)$. Next, we will show that the total time complexity for solving subproblems $(v, q, \vec{a})$ with v not being a rich vertex is also $O(kn)$. For each vertex v, let H'_v denote the total time spent for solving subproblems $(u, q, \vec{a})$ with $u \in T_v$ being not rich, and define $H_v := nH'_v$. If v is rich, then $H'_v = H'_{u_1} + H'_{u_2}$, which implies $H_v = H_{u_1} + H_{u_2}$. Otherwise, v is not rich, and then, it can be seen that $H'_v \leq H'_{u_1} + H'_{u_2} + c \min\{|V(T_{u_1})|, k/2\} \min\{|V(T_{u_2})|, k/2\}$, where c is a constant, and $V(T_{u_j})$ indicates the set of vertices in T_{u_j} for $j \in \{1, 2\}$. This implies that $H_v \leq H_{u_1} + H_{u_2} + cn \min\{|V(T_{u_1})|, k/2\} \min\{|V(T_{u_2})|, k/2\}$. By Lemma 2 in [71], we have $H_v \leq ckn|V(T_v)|$, where $V(T_v)$ indicates the set of vertices in T_v. Thus, we obtain that $H'_r \leq H_r/n \leq ckn|V|/n = ckn$. Hence, the total time complexity of the dynamic programming algorithm is $O(kn)$. □

3.4 Summary

In this chapter, we have proved that the new proposed approximation algorithm achieves a tight approximation ratio of $(2 - 1/|J|)$ for the k-median Steiner forest problem that jointly optimizes facility locations and network connections, which is better and much simpler to prove than the existing 2-approximation algorithm in the literature. Computational experiments and examples show that our new algorithm can always produce solutions of equal or better quality than the existing algorithms, up to 50% improvement in some cases. Moreover, we have further considered two special cases of the problem, where either each vertex containers a client, or all the vertices are located in a tree-shaped network. For these two cases, new polynomial time algorithms, which can produce optimal solution, have been proposed. In the future, one possible research direction is to improve the worst-cast approximation ratio for the problem, or to develop approximation algorithms with constant ratio to other variants of the problem.

CHAPTER 4

Conclusions

This thesis comprises two essays, each of which has studied a different logistics optimization problem that has wide applications within the transportation industry. For these two problems, we have developed new improved approximation algorithms with constant approximation ratios. Moreover, for some special cases which are commonly seen in practice, we have also proposed new polynomial time algorithms that can solve them to optimality. In the following, we summarize, respectively, the main results of each essay, and provide several possible interesting directions for future research.

In the first essay, for the CTSPPD-T problem, we developed a 2-approximation algorithm that has a polynomial time complexity provided the ratio $(\sum_{v\in V}|q(v)|)/k$ is polynomially bounded by $|V|$. Since each pickup or delivery point v needs at least $\lceil|q(v)|/k\rceil$ times of pickups or deliveries, every feasible route must consist of at least $\sum_{v\in V}\lceil|q(v)|/k\rceil$ states. Thus, unless it requires an exponential number of pickups or deliveries, which is unlikely in practice, the ratio $(\sum_{v\in V}|q(v)|)/k$ is always polynomially bounded by $|V|$. Moreover, computational results show that our algorithm also achieves good average performance over randomly generated instances, and exhibits a much shorter running time and better solution quality than a greedy algorithm.

Note that in keeping with the literature [2, 18, 37], in this essay we only consider

the case where the request for pickups and deliveries is balanced, with $\sum_{v \in V} q(v) = 0$. Therefore, one natural direction of future research is to consider an unbalanced case $\sum_{v \in V} q(v) > 0$, where the vehicle may only serve partial pickup requests or partial delivery requests due to its unbalanced nature. One possible method is to firstly determine an actual pickup or delivery amount $q'(v)$ for each $v \in V$, so as to obtain a balanced instance, with $\sum_{v \in V} q'(v) = 0$, to which therefore our newly developed algorithm can be applied. The resulting route will achieve an approximation ratio of 2 if, similar to (2.2), $\sum_{u \in V} 2n'(u)d(e(u))$ is a lower bound on the optimal route length, where $n'(u) = \max\{\lceil |q'(T_u)|/k \rceil, 1\}$. To achieve this, we can determine $q'(v)$ for $v \in V$ so as to minimize $\sum_{u \in V} 2n'(u)d(e(u))$. Therefore, using the above mentioned method of determining an actual amount, seeing whether the final solution obtained has a better quality performance and whether it has a polynomial time complexity are interesting problems for future research to consider.

Another direction for our future research is to consider a more complicated setting for the problem, such as there being multiple types of products to transport, as well as being able to use multiple vehicles, since in actual practice there always exist multiple types of products to transport or carriers always using a number of vehicles in their practice operations. For such cases, the results obtained in this essay can be further extended and embedded into more sophisticated methods, e.g., as a method of generating routes used as columns in column generation or as initial solutions in meta-heuristics, such as simulated annealing or local search. This can be possible in order to solve large scale instances and let the problem be of more practical value.

In the second essay, we have presented a new approximation algorithm for the k-median Steiner forest problem that jointly optimizes facility locations and network connections. The new algorithm is based on a simple transformation from a minimum spanning tree of the clients and a new vertex that replaces all the facilities. Compared with the existing best 2-approximation algorithm that combines a Lagrangian relax-

ation with a primal-dual schema, our new algorithm is much simpler, and achieves a better approximation ratio that is easier to be proved. We have also shown that the new algorithm can always produce solutions of equal or better quality than the existing 2-approximation algorithm, and the quality improvement can be up to 50% in some cases. Moreover, we have developed new polynomial time algorithms that can solve the problems to optimality for two special cases, where either each vertex contains a client, or all the vertices are located in a tree-shaped network.

One direction of our future research is to improve the approximation ratio for the k-median Steiner forest problem. For this, one possible approach is to extend the techniques that have successfully been used in improving the approximations of the Steiner tree problem, for which the current best approximation ratio is $\ln(4)+\varepsilon < 1.39$ [14].

The other direction of our future research is to develop constant ratio approximation algorithms for other variants of the problem. For example, one variant is that with a capacity constraint that restricts the maximum number of clients that can be connected to each facility. The constraint is similar to that in the classical capacitated facility location problem, which has wide applications in practice and is well-studied in the literature [8, 40, 80]. Moreover, another possible variant is to extend to the case not only having the facility number constraints, but also having a different facility opening cost, i.e., each facility has a different fixed cost to open [75]. A similar problem, k-facility location problem, has been considered in the literature [41, 79], and can be applied to a more general setting in practice. For these, the results obtained in this study have laid down a sound foundation that can be extended upon even further.

APPENDICES

APPENDIX A

Formulation for k-median Steiner forest problem

A.1 Mixed integer programming formulation

We now propose a multi-commodity flow formulation for the k-median Steiner forest problem. As mentioned before in Section 3.2.3.1, the problem on $G = (V, E)$ can be formulated as a constrained Steiner tree problem on a new undirected graph $\hat{G}$ with a root degree constraint [29, 33], where the vertex set consists of V and a new root vertex r, and the edge set consists of E and the root edges (r, v) for $v \in W$ with $\ell(r, v) = 0$, denoted by E_r. Hence, $\hat{G} = (V \cup \{r\}, E \cup E_r)$.

For each client $j \in J$, define a commodity j. Given the graph $\hat{G}$, define a bi-directed graph $B = (V \cup \{r\}, A)$ through bidirecting every edge of E and adding the root arcs (r, v) for $v \in W$. For each arc $a = (u, v) \in A$, let f_{uv}^j be the flow variable representing the commodity j flows from node u to node v. For each edge $e \in E \cup E_r$, let x_e be a binary variable equal to 1 if and only if the edge e is in the Steiner tree (0 otherwise). Therefore, the multi-commodity flow formulation is the following:

$$\min \sum_{e \in E} \ell(e) x_e \tag{A.1}$$

subject to:

$$\sum_{e \in E_r} x_e \leq k \tag{A.2}$$

$$\sum_{u \in V} f_{uv}^j - \sum_{u \in V} f_{vu}^j = \begin{cases} -1 & \text{for } v = r,\ j \in J \\ 1 & \text{for } v = j,\ j \in J \\ 0 & \text{for } v \in V \setminus \{r, j\},\ j \in J \end{cases} \tag{A.3}$$

$$f_{uv}^j \leq x_e, \text{ for each } e \in E \cup E_r,\ (u,v) \in A \text{ with } e = (u,v),\ j \in J \tag{A.4}$$

$$f_{uv}^j \geq 0, \text{ for each } (u,v) \in A,\ j \in J \tag{A.5}$$

$$x_e \in \{0,1\}, \text{ for each } e \in E \cup E_r \tag{A.6}$$

The objective function (A.1) asks for minimizing the total connection cost in the tree. Constraint (A.2) ensures the degree of vertex r not exceeding k. Constraints (A.3) are the flow balancing constraints for each commodity j that ensure only one unit outflow of vertex r and one unit inflow of client j, as well as the outflow equals to inflow in other vertices. Moreover, constraints (A.4) impose that there are flows in the arc (u,v) if and only if the edge e is included in the Steiner tree, i.e., $x_e = 1$. Finally, constraints (A.5) and (A.6) define f_{uv}^j as non-negative continuous variables and x_e as binary variables.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] R. Agarwal and O. Ergun. Ship scheduling and network design for cargo routing in liner shipping. *Transportation Science*, 42(2):175–196, 2008.

[2] S. Anily and J. Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics*, 46(6): 654–670, 1999.

[3] S. Anily and R. Hassin. The swapping problem. *Networks*, 22(4):419–433, 1992.

[4] S. Anily and G. Mosheiov. The traveling salesman problem with delivery and backhauls. *Operations Research Letters*, 16(1):11–18, 1994.

[5] C. Archetti and M. G. Speranza. Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1-2):3C–22, 2012.

[6] C. Archetti, M. W. P. Savelsbergh, and M. G. Speranza. Worst-case analysis for split delivery vehicle routing problems. *Transportation science*, 40(2):226–234, 2006.

[7] E. M. Arkin, R. Hassin, and L. Klein. Restricted delivery problems on a network. *Networks*, 29(4):205–216, 1997.

[8] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[9] 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.

[10] 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.

[11] A. Balakrishnan, M. Banciu, K. Glowacka, and P. Mirchandani. Hierarchical approach for survivable network design. *European Journal of Operational Research*, 225(2):223 – 235, 2013.

[12] S. H. Bartholomew. *Estimating and bidding for heavy construction*. Prentice-Hall, Englewood Cliffs, NJ, 2000.

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

[14] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM*, 60(1):6:1–6:33, 2013.

[15] J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, to appear*. SIAM, 2015.

[16] T. Carnes and D. B. Shmoys. Primal-dual schema and Lagrangian relaxation for the k-location-routing problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 99–110. Springer, 2011.

[17] D. O. Casco, B. L. Golden, and E. A. Wasil. *Vehicle routing: methods and studies*, chapter Vehicle routing with backhauls: Models, algorithms, and case studies, pages 127–147. North-Holland, Amsterdam, 1988.

[18] P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28(6):2133–2149, 1999.

[19] S. Chamberland, B. Sanso, and O. Marcotte. Topological design of two-level telecommunication networks with modular switches. *Operations Research*, 48 (5):745–760, 2000.

[20] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM, 1999.

[21] B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

[22] H. Chen, A. M. Campbell, and B. W. Thomas. Network design for time-constrained delivery. *Naval Research Logistics*, 55(6):493–515, 2008.

[23] S. Chen, B. Golden, and E. Wasil. The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results. *Networks*, 49(4):318–329, 2007.

[24] D. Chhajed and T. J. Lowe. An $O(n)$ algorithm for a special case of the multi-median location problem on a tree. *European Journal of Operational Research*, 63(2):222–230, 1992.

[25] M. Christiansen, K. Fagerholt, B. Nygreen, and D. Ronen. Maritime transportation. *Handbooks in operations research and management science*, 14:189–284, 2007.

[26] M. Christiansen, K. Fagerholt, B. Nygreen, and D. Ronen. Ship routing and scheduling in the new millennium. *European Journal of Operational Research*, 228(3):467–483, 2013.

[27] I. Contreras and E. Fernández. General network design: A unified view of combined location and network design problems. *European Journal of Operational Research*, 219(3):680–697, 2012.

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (2nd Edition)*, chapter Binary Search Trees, page 254. The MIT Press, 2001.

[29] D.-Z. Du and X. Cheng. *Steiner Trees In Industries*. Kluwer Academic Publisher, 2000.

[30] R. Epstein, A. Neely, A. Weintraub, F. Valenzuela, S. Hurtado, G. Gonzalez, A. Beiza, M. Naveas, F. Infante, F.Alarcon, G. Angulo, C. Berner, J. Catalan, C. Gonzalez, and D. Yung. A strategic empty container logistics optimization in a major shipping company. *Interfaces*, 42(1):5–16, 2012.

[31] H. N. Gabow and R. E. Tarjan. Efficient algorithms for a family of matroid intersection problems. *Journal of Algorithms*, 5(1):80–131, 1984.

[32] M. R. Garey and D. S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[33] M. X. Geomans and Y. Myung. A catalog of steiner tree formulations. *Networks*, 23:19–28, 1993.

[34] É. Gourdin, M. Labbé, and G. Laporte. The uncapacitated facility location problem with client matching. *Operations Research*, 48(5):671–685, 2000.

[35] D. Henderson, D. E. Vaughan, S. H. Jacobson, R. R. Wakefield, and E. C. Sewell. Solving the shortest route cut and fill problem using simulated annealing. *European Journal of Operational Research*, 145(1):72–84, 2003.

[36] H. Hernández-Pérez and J. J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139, 2004.

[37] H. Hernández-Pérez and J. J. Salazar-González. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38(2):245–255, 2004.

[38] H. Hernández-Pérez and J. J. Salazar-González. The one-commodity pickup-and-delivery traveling salesman problem: inequalities and algorithms. *Networks*, 50(4):258–272, 2007.

[39] H. Hernández-Pérez and J. J. Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995, 2009.

[40] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, pages 731–740. ACM, 2002.

[41] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM*, 50(6):795–824, 2003.

[42] K. l. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[43] H. Jiang and C. Barnhart. Dynamic airline scheduling. *Transportation Science*, 43(3):336–354, 2009.

[44] M. Karpinski and A. Zelikovsky. New approximation algorithms for the Steiner tree problems. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997.

[45] 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.

[46] 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.

[47] N. Katoh and T. Yano. An approximation algorithm for the pickup and delivery vehicle routing problem on trees. *Discrete Applied Mathematics*, 154(16):2335–2349, 2006.

[48] H. L. M. Kerivin, M. Lacroix, and A. R. Mahjoub. Models for the single-vehicle preemptive pickup and delivery problem. *Journal of Combinatorial Optimization*, 23(2):196–223, 2012.

[49] D. Klabjan, E. L. Johnson, G. L. Nemhauser, E. Gelman, and S. Ramaswamy. Airline crew scheduling with regularity. *Transportation Science*, 35(4):359–374, 2001.

[50] B. H. Korte and J. Vygen. *Combinatorial optimization: theory and algorithms*, chapter 13. Matroids, pages 305–336. Springer Verlag, 2008.

[51] M. Labbe, G. Laporte, and H. Mercure. Capacitated vehicle routing problems on trees. *Operations Research*, 39(2):616–622, 1991.

[52] X. Lai and Z. Xu. Improved algorithms for joint optimization of facility locations and network connections, 2015. Forthcoming.

[53] S. Li and O. Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the Forty-Fifth annual ACM Symposium on Theory of Computing*, pages 901–910. ACM, 2013.

[54] A. Lim, B. Rodrigues, and J. Zhang. Tabu search embedded simulated annealing for the shortest route cut and fill problem. *Journal of the Operational Research Society*, 56(7):816–824, 2004.

[55] F. Louveaux and J. J. Salazar-González. On the one-commodity pickup-and-delivery traveling salesman problem with stochastic demands. *Mathematical Programming*, 119(1):169–194, 2009.

[56] 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.

[57] G. Mosheiov. The travelling salesman problem with pick-up and delivery. *European Journal of Operational Research*, 79(2):299–310, 1994.

[58] P. A. Mullaseril, M. Dror, and J. Leung. Split-delivery routeing heuristics in livestock feed distribution. *Journal of the Operational Research Society*, 48(2): 107–116, 1997.

[59] M. Nowak, O. Ergun, and C. C. White. Pickup and delivery with split loads. *Transportation Science*, 42(1):32–43, 2008.

[60] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[61] C. Prodhon and C. Prins. A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238(1):1–17, 2014.

[62] H. J. Prömel and A. Steger. A new approximation algorithm for the Steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36(1):89–101, 2000.

[63] R. Ravi. A primal-dual approximation algorithm for the Steiner forest problem. *Information processing letters*, 50(4):185–189, 1994.

[64] R. Ravi and A. Sinha. Approximation algorithms for problems combining facility location and network design. *Operations Research*, 54(1):73–81, 2006.

[65] M. G. C. Resende and P. Pardalos. *Handbook of Optimization in Telecommunications*. Springer, 2006.

[66] H. Richter. Thirty years of airline operations research. *Interfaces*, 19(4):3–9, 1989.

[67] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779. SIAM, 2000.

[68] M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[69] G. Sierksma and G. A. Tijssen. Routing helicopters for crew exchanges on offshore locations. *Annals of Operations Research*, 76:261–286, 1998.

[70] S. H. Song, K. S. Lee, and G. S. Kim. A practical approach to solving a newspaper logistics problem using a digital map. *Computers & industrial engineering*, 43 (1-2):315–330, 2002.

[71] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996.

[72] J. N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992.

[73] V. V. Vazirani. *Approximation algorithms*, chapter 3. Steiner Tree and TSP, pages 27–37. Springer, 2001.

[74] F. Wang, A. Lim, and Z. Xu. The one-commodity pickup and delivery travelling salesman problem on a path or a tree. *Networks*, 48(1):24–35, 2006.

[75] F. Wang, X. Lai, and N. Shi. A multi-objective optimization for green supply chain network design. *Decision Support Systems*, 51(2):262–269, 2011.

[76] L. Xu, Z. Xu, and D. Xu. Exact and approximation algorithms for the min–max k-traveling salesmen problem on a tree. *European Journal of Operational Research*, 227(2):284–292, 2013.

[77] Z. Xu, X. Lai, A. Lim, and F. Wang. An improved approximation algorithm for the capacitated TSP with pickup and delivery on a tree. *Networks*, 63(2): 179–195, 2014.

[78] A. Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[79] P. Zhang. A new approximation algorithm for the k-facility location problem. *Theoretical Computer Science*, 384:126–135, 2007.

[80] Z. Zhang, G. Berenguer, and Z. M. Shen. A capacitated facility location model with bidirectional flows. *Transportation Science*, 49(1):114–129, 2015.