Approximation Schemes for Capacitated Vehicle Routing on Graphs of Bounded Treewidth, Bounded Doubling, or Highway Dimension

by

Aditya Jayaprakash

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In this thesis, we present Approximation Schemes for Capacitated Vehicle Routing Problem (CVRP) on several classes of graphs. In CVRP, introduced by Dantzig and Ramser in 1959 [16], we are given a graph $G = (V, E)$ with metric edges costs, a depot $r \in V$, and a vehicle of bounded capacity Q. The goal is to find a minimum cost collection of tours for the vehicle that return to the depot, each visiting at most Q nodes, such that they cover all the nodes. This generalizes the classic travelling salesman problem (TSP) and has been studied extensively. In the more general setting, each node v has a demand d_v and the total demand of each tour must be no more than Q. Either the demand of each node must be served by one tour (unsplittable) or can be served by multiple tour (splittable). The best known approximation algorithm for general graphs has ratio $\alpha + 2(1 - \epsilon)$ (for the unsplittable) and $\alpha + 1 - \epsilon$ (for the splittable) for some fixed $\epsilon > \frac{1}{3000}$, where α is the best approximation for TSP. Even for the case of trees, the best approximation ratio is 4/3 [6] and it has been an open question if there is an approximation scheme for this simple class of graphs. Das and Mathieu [17] presented an approximation scheme with time $n^{\log^{O(1/\epsilon)} n}$ for Euclidean plane $\mathbb{R}^2$. No other approximation scheme is known for any other class of metrics (without further restrictions on Q). In this thesis, we make significant progress on this classic problem by presenting Quasi-Polynomial Time Approximation Schemes (QPTAS) for graphs of bounded treewidth, graphs of bounded highway dimensions, and graphs of bounded doubling dimensions. For comparison, our result implies

an approximation scheme for Euclidean plane with run time $n^{O(\log^6 n/\epsilon^5)}$.

To Amma and Appa

You take 100% of the shots you miss.

– Wayne Gretzky.

Acknowledgements

First and foremost, I would like to thank my advisor, Mohammad R Salavatipour for his support during my graduate program. Mohammad has always patiently guided me throughout this thesis, and I am very fortunate to have received his mentorship. He is an all-around fantastic and everything I could hope for in an advisor.

I would like to thank Zachary Friggstad and Guohui Lin for serving on my thesis committee. Their feedback was instrumental in improving this thesis. I am grateful to have worked and learned from Zach during the group project.

My graduate school experience would not have been the same without my Theory group lab mates - Ramin Mousavi, Haozhou Pang, Brandon Fuller and Maryam Mahboub. I would like to thank them for making the lab a fun place to work. Ramin has been a great friend and I would like to thank Ramin for his help and advice when I started at Alberta. I would like to thank Brandon for being a great friend and for going above and beyond to help me when I was job searching.

I am very happy to have met Archit Sakhadeo, Taivanbat (TK) Badamdorj, Shivam Garg, Parash Rahman and Soumyadeep Pal during my time at Alberta. Archit has been a great friend, and I have thoroughly enjoyed our conversations about life, politics, and food. Archit is someone I can always rely on for advice. TK was an amazing roommate and I am fortunate to have shared his company during the pandemic. I will miss our conversations about all-things-random at the kitchen. Shivam, Parash and Soumyadeep were won-

derful to hangout with and codenames was always fun with their company.

Life would not be the same without my friends from undergrad in Waterloo - Milap Sheth, Yaron Koller, Aayush Shah, Nihal Pednekar and Rwitaban (Ray) Banerjee. Milap has always been a great friend, and has helped me get through many difficult times - he is like an older brother to me, and I very grateful to him. Yaron is an all-around fantastic friend, and I would like to thank him in particular for picking me up at the airport in February, 2020. I met Aayush during an intership early in my undergrad and he has been an incredible friend ever since, and someone I can always count on. I would like to thank Nihal for being a great friend, and for helping me through my job-search phase. I have enjoyed our long conversations very much. I would like to thank Ray for the banter and for his friendship during our trips to Japan and Montreal.

Back to my undergraduate days at Waterloo - I would not have gone to graduate school without the help and support of Naomi Nishimura. I would like to thank her immensely for giving me a chance to work with her. I got excited about theoretical computer science only after taking a course on the theory of computation taught by Eric Blais. His enthusiasm for theory was contagious, and I would like to thank him for that. I would like to thank Lap Chi Lau for being a great instructor. I learned a tremendous amount from his graduate course on randomized algorithms and it prepared me well for graduate school. Going back even further, I would like to thank my high-school mathematics tutor B. Rajkumar for keeping me excited about math.

I would like to thank Nivasini Ananthakrishnan for her constant support. Her presence in my life makes everything more enjoyable; she helped me get through the pandemic. I would like to thank my parents - Jayaprakash and Duniya. They have always been my backbone and have supported me throughout, in numerous ways. They always encouraged me to pursue my interests.

This thesis is dedicated to them. Life would not be the same without my cat, Misty. She is a constant source of joy and love and makes me smile throughout the day. Adopting her was one of the best decisions I ever made and I am grateful to her.

Contents

List of Figures

Chapter 1

Introduction

Numerous problems that arise in industrial and consumer applications can be viewed as an instance of the problem of routing a vehicle along a road network to provide a service at different locations. We refer to these problems as *vehicle routing problems*. Some of the well-known vehicle routing problems are:

- The Travelling Salesman Problem (TSP): Given a set of points and a starting point, find the shortest route that visits every location.
- The Capacitated Vehicle Routing Problem (CVRP): Given a set of points, a depot, and a vehicle with capacity Q, find a minimum length collection of tours, each starting from the depot and visiting at most Q customers, and whose union covers all the customers.
- The Orienteering Problem: Find a route of bounded length that reaches as many locations as possible.
- The Travelling Repairman Problem: Find a route that reaches every location, minimizing the average wait time.
- The School Bus Problem: Find the smallest number of tours such that every location is visited within a given time.

The Travelling Salesman Problem (TSP) appears in industrial applications, where we wish to minimize the amount of time a tool or machine spends moving between its required positions. In 1962, Proctor & Gamble famously ran a contest, offering $10,000 for the shortest tour through 33 cities in the US

[1][2]. The Capacitated Vehicle Routing Problem (CVRP), Orienteering Problem and Travelling Repairman Problems commonly appear in scheduling package deliveries.

All the problems mentioned above are in general **NP**-Hard. In this thesis, we consider algorithms that produce an *approximate* solution; that is, the solution found is at most α times worse than the exact (optimal) value for either some $\alpha \in \mathbb{R}^+ > 0$ or α, a function of the size of the input instance. Although we generally like to find polynomial time algorithms, we sometimes settle for algorithms that run in time $n^{O(\log^c n)}$ for some constant $c > 0$. We refer to an algorithm that run in time $n^{O(\log^c n)}$ as a quasi-polynomial time algorithm where n is the size of the instance. It is believed that **NP**-Hard problems do not have quasi-polynomial algorithms. Also, a quasi-polynomial time approximation algorithm suggests that it might be possible to find similar approximation algorithms in polynomial time.

The Capacitated Vehicle Routing Problem is **APX**-hard in general metric spaces, meaning that unless $\mathbf{P} = \mathbf{NP}$, the problem would not have a $(1 + \epsilon_0)$-approximation, for some fixed $\epsilon_0 > 0$. So a natural research focus has been on structured metric spaces. We will give approximation algorithms for the Capacitated Vehicle Routing Problem in various graph classes. We first begin with an introduction to the terminology and concepts used in this thesis, followed by an overview of the problems we consider. We will then discuss related work in the literature, and the results we obtain.

1.1 Preliminaries

We start by formalizing the terminology used throughout this thesis. The definitions given here are adapted from [31], [29], [14], and [32].

[1] `http://www.math.uwaterloo.ca/tsp/us/history.html`
[2] `http://www.math.uwaterloo.ca/tsp/us/img/car54.jpg`

1.1.1 Graphs and Metrics

Graphs. We only consider simple graphs in this thesis, and use the term *graph* in this thesis to mean an undirected graph. A graph G is defined by its edge set $E(G) = \{e_1, e_2, \ldots, e_m\}$ and vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$, where each edge $e \in E(G)$ is an unordered pair of vertices in $V(G)$. To simplify notation, we will drop the parameters $V(G)$ and $E(G)$ where the graph is clear from context and denote $G = (V, E)$.

For an edge $e = uv \in E(G)$, we say u and v are *adjacent* and e is *incident* to u and v. The *neighbours* of a vertex v is the set of vertices u such that u and v are adjacent, denoted by $N_G(v)$, or simply $N(v)$ when G is clear from context.

A *subgraph* of a graph G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We write $H \subseteq G$ and say that G contains H.

A *path* is a simple graph whose vertices can be ordered so that two vertices are adjacent if and only if they are consecutive in the list. A *cycle* is a graph with an equal number of vertices and edges whose vertices can be placed around a circle so that two vertices are adjacent if and only if they appear consecutively along the circle. A *walk* is a list $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges such that, for $1 \leq i \leq k$, the edge e_i has endpoints v_{i-1} and v_i. An *acyclic graph* is a graph that does not contain any cycle as a subgraph. A *connected graph* is a graph G where for every pair of vertices $u, v \in V(G)$, there is a path in G from u to v. A *tree* is an acyclic, connected graph. A *complete graph* is a graph whose vertices are pairwise-adjacent.

A *tree decomposition* of a graph G is a pair $(T, \{B_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $B_t \subseteq V(G)$, called a bag, such that the following three conditions hold:

1. $\cup_{t \in V(T)} B_t = V(G)$. In other words, every vertex of G is in at least one

bag.

2. For every $uv \in E(G)$, there exists a node t of T such that bag B_t contains both u and v.

3. For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in B_t\}$, i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree of T.

The *width* of a tree decomposition is one less than the maximum bag size of that decomposition. The *treewidth* of a graph G is the minimum k such that there exists a tree decomposition T of G with bags of size at most $k+1$. Note that trees have treewidth 1.

Weighted graphs and metrics. A *weighted graph* is a graph with numerical edge labels. We will assume throughout this thesis that these labels are non-negative, and refer to them as edge costs or weights, denoted by $w(e)$ or w_{uv}. A *metric space* (X, d_X) consists of a set of points X and a distance function $d_X : X \times X \to R^{\geq 0}$ which satisfies the following properties:

1. For every $x, y \in X$, $d_X(x, y) \geq 0$.

2. For every $x \in X$, $d_X(x, x) = 0$.

3. For every $x, y \in X$, $d_X(x, y) = d_X(y, x)$. This property is referred to as symmetry.

4. For every $x, y, z \in X$, $d_X(x, z) \leq d_X(x, y) + d_X(y, z)$. This is referred to as triangle inequality.

Metrics can be equivalently defined as a complete weighted graph whose edge costs satisfy the triangle inequality. Any metric (V, d) can be converted into a graph G by letting $V(G) = V$, and $w_{uv} = d_G(u, v)$. We define the distance $d_G(u, v)$ between two vertices u and v in a graph G as the minimum cost of a u-v path in G; if no such path exists, the distance is undefined. Let $d_{\max} = \max_{u,v \in V(G), u \neq v} d_G(u, v)$, $d_{\min} = \min_{u,v \in V(G), u \neq v} d_G(u, v)$ and we will denote the

aspect ratio by $\Delta = \frac{d_{\max}}{d_{\min}}$.

For a metric space (X, d), the (closed) *ball* of radius r around a point $x \in X$ is $B_r(x) = \{y \in X : d(y, x) \leq r\}$. We say $Y \subseteq X$ is *bounded* if it is contained within a ball of finite radius. The *doubling dimension* of X is the smallest $k \in \mathbb{R}$ such that for any given $x \in X, r > 0$, we have $B_r(x) \subseteq \cup_{y \in Y} B_{r/2}(y)$ for some $Y \subseteq X$ satisfying $|Y| \leq 2^k$. The doubling dimension is a measure of how fast the volume of a metric space is growing.

Abraham et al [1], [2] formally defined the notion of *highway dimension* to model transportation networks. The *highway dimension* of a graph G is the smallest integer k such that, for some universal constant $c \geq 4$, for every $r \in \mathbb{R}^+, v \in V(G)$, and every ball $B_{cr}(v)$ of radius cr, there are at most k vertices in $B_{cr}(v)$ hitting all shortest paths in $B_{cr}(v)$ of length more than r.

As noted in [18], Abraham et al [2] chose $c = 4$ and notes that this choice is somewhat arbitrary. Similar to the work of [18], we will allow the flexibility of being able to choose a slightly larger value of c. We will let $\lambda = c - 4$ and call it the *violation*.

1.1.2 Optimization Problems and Approximation Algorithms

Decision problem and NP. A *decision problem* is a problem that can be answered with either "yes" or "no". We view decision problems as languages over the binary alphabet $\{0, 1\}^*$; the language L corresponding to some decision problem is the set of all strings that encode a "yes" to the problem.

A Turing machine M is a polynomial-time verifier if for all input pairs $(x, y) \in \{0, 1\}^*$ it reads, M halts after $p(|x|)$ steps where p is a polynomial function. A language is decidable by M if for all $x \in \{0, 1\}^*$, the following holds,

- If $x \in L$, there exists $y \in \{0,1\}^{p(|x|)}$ such that M accepts (x, y).
- If $x \notin L$, for every $y \in \{0,1\}^*$, M rejects (x, y).

The class **NP** is all languages L decided by a polynomial-time verifier. Let L_1 and L_2 be two languages in **NP**. L_1 reduces to L_2 if there is a Turing machine that, given a string $x \in \{0,1\}^*$, outputs a string $y \in L_2$ if and only if $x \in L_1$ and does so in poly($|x|$) steps. A language L is **NP**-Hard if for every language $L' \in \mathbf{NP}$, L' reduces to L. A language is **NP**-*complete* if both L is **NP**-Hard and $L \in \mathbf{NP}$.

Optimization problems. The *size* of an instance I, written $|I|$ is the number of bits needed to describe it. An **NP**-*optimization problem* Π consists of:

- A set of valid *instances* D_Π, where we can determine if some instance $I \in D_\Pi$ in time poly($|I|$). We assume all instances $I \in D_\Pi$ can be expressed as finite binary strings; this implies that all numeric values must be integer or rational.
- A set of *feasible solutions* $S_\Pi(I)$ for each instance $I \in D_\Pi$, where we can determine is $s \in S_\Pi(I)$ in time poly($|I|$). The length of each solution must be poly($|I|$).
- An *objective function* obj_Π that assigns each instance-solution pair (I, s) a non-negative value, computable in time poly($|I|$).

We also specify whether Π is a *minimization problem* or a *maximization problem*. For a minimization/maximization problem Π and instances $I \in D_\Pi$, an *optimal solution* is a feasible solution $s \in S_\Pi(I)$ that minimizes/maximizes the value of obj_Π; that is, $\operatorname{argmin}_{s \in S_\Pi} \text{obj}_\Pi(I, s)$ or $\operatorname{argmax}_{s \in S_\Pi} \text{obj}_\Pi(I, s)$. We denote such a solution as $\text{OPT}_\Pi(I)$, or simply OPT if the problem and instance are clear from the context. The objective value of optimum solution is denoted by OPT.

An **NP** optimization problem Π gives rise to a class of **NP** decision problems, by asking if a feasible solution of at most or at least some objective

value exists (for minimization/maximization problems).

For the optimization problems we consider in this thesis, the decision versions have been shown to be **NP**-Hard, and so we say that the optimization problems are also **NP**-Hard. Unless $\mathbf{P} = \mathbf{NP}$, there is no efficient algorithm that be used to solve an **NP**-Hard problem exactly. A different strategy is to find a time-efficient algorithm that returns a solution that is never more than a given factor worse than the optimal solution. We call these *approximation algorithms*.

Approximation algorithms. Let Π be a minimization (maximization) problem, and let $\alpha : \mathbb{Z}^+ \to \mathbb{Q}^+$ be a function such that $\alpha \geq 1$ for all inputs. An algorithm $\mathcal{A}$ is an α-approximation for Π if, for all instances I, $\mathcal{A}$ returns a feasible solution $s \in S_\Pi(I)$ such that $\text{obj}_\Pi(I, s) \leq \alpha(|I|) \cdot \text{OPT}_\Pi(I)$, and the running time is bounded by $\text{poly}(|I|)$. The function α is called the *approximation ratio* of $\mathcal{A}$.

It is sometimes difficult to obtain an algorithm that meets this definition exactly. We might need to relax the running time constraint, for example to a quasi-polynomial factor. Or, the algorithm makes random choices, and so the approximation ratio only holds in expectation over all random choices. We still refer to these as approximation algorithms, although we will state such relaxation explicitly.

An algorithm $\mathcal{A}$ is an *approximation scheme* for the minimization (maximization) problem Π if for the valid instance I and error parameter $\epsilon > 0$, it returns a feasible solution s such that $\text{obj}_\Pi(I, s) \leq (1 + \epsilon)\text{OPT}_\Pi(I)$ (or $\text{obj}_\Pi(I, s) \geq (1 - \epsilon)\text{OPT}_\Pi(I)$ if it is a maximization problem). We call $\mathcal{A}$ to be a *polynomial time approximation scheme* (PTAS) if its running time is $\text{poly}(|I|)$ for each fixed ϵ. We call $\mathcal{A}$ a *fully polynomial time approximation scheme* (FPTAS) if its running time is $\text{poly}(|I|, 1/\epsilon)$ for each fixed ϵ. We call $\mathcal{A}$ a *quasi-polynomial time approximation scheme* (QPTAS) if its running time

is $|I|^{O(\log^c |I|)}$ for each fixed ϵ for some absolute constant $c > 0$.

Let Π and Π' be two optimization problems. Π PTAS-*reduces* to Π' if there exists an algorithm $\mathcal{A}$ and function $c : \mathbb{R}^+ \to \mathbb{R}$, where for each valid instance I of Π and for each fixed $\epsilon > 0$,

- Algorithm $\mathcal{A}$ returns an instance $I' = \mathcal{A}(I, \epsilon)$ of Π' in time poly($|I|$), such that if I is feasible, then I' is feasible.
- Given any feasible solution $s' \in S_{\Pi'}(I')$, there exists a feasible solution $s \in S_\Pi(I)$ such that if $\text{obj}_{\Pi'}(I', s') \leq (1 + c(\epsilon))\text{OPT}_{\Pi'}(I')$, then $\text{obj}_\Pi(I, s) \leq (1 + \epsilon) \cdot \text{OPT}_\Pi(I)$.

The problem Π is said to be in class **APX** if it admits any constant approximation. An optimization problem Π is said to be **APX**-hard if for every other problem $\Pi' \in$ **APX**, Π' PTAS-reduces to Π. If in addition $\Pi \in$ **APX**, then Π is said to be **APX**-*complete*.

Let L be a language in **NP**, and Π be a minimization (maximization) problem. Let $g : \{0, 1\}^* \to D_\Pi$ be a function computable in polynomial time that maps yes and no instances of L to instances of Π. We say g is a gap-introducing reduction from L to instances of Π if a polynomial-time computable function $h : D_\Pi \to \mathbb{R}^+$ and constant α exist where

- If x is a yes-instance of L, then for a minimization problem, $\text{OPT}_\Pi(g(x)) \leq h(g(x))$ and for a maximization problem, $\text{OPT}_\Pi(g(x)) \geq h(g(x))$, and
- If x is a no-instance of L, then for a minimization problem $\text{OPT}_\Pi(g(x)) > \alpha h(g(x))$ and for a maximization problem, $\text{OPT}_\Pi(g(x)) < h(g(x))/\alpha$.

The constant α is called the gap size.

Hardness of approximation. A *hardness proof* shows that a certain optimization problem cannot be approximated better than some threshold assuming certain complexity theoretic assumption. It was shown that approximating

MAX-3SAT better than $(1+\epsilon_0)$ for some absolute $\epsilon_0 > 0$ is **NP**-Hard, ruling out a PTAS assuming $\mathbf{P} \neq \mathbf{NP}$ [29]. Since this problem is also **APX**-*complete*, a consequence of this is that any **APX**-hard optimization problem Π does not have a PTAS unless $\mathbf{P} = \mathbf{NP}$.

1.1.3 Metric Embeddings

The definitions given here are adapted from [26]. The main motivation behind metric embedding is to map a complicated metric space into a simpler metric space which is easier to work with. A common theme in devising an approximation algorithm for general metrics is by mapping shortest path metrics on general graphs to tree metrics and being able to use algorithms that work on trees (but not on general graphs).

Ideally, the mapping we consider needs to preserve the structure of the original metric space. The ideal mapping are the following form.

Definition 1 (Isometric Embedding) *A mapping $f : X \to Y$ of a metric space (X, d_X) to a metric space (Y, d_Y) is an* isometric embedding *if for every two points $x_1, x_2 \in X$, we have*

$$d_Y(f(x_1), f(x_2)) = d_X(x_1, x_2)$$

Isometric embeddings are those that preserve the distances between points exactly. However, in many cases, isometric embeddings do not exist. Instead, there is a notion of an approximate embedding which is captured by the following definition that allows the distances to change slightly while still approximately preserving the structure of the original metric space.

Definition 2 (Distortion of an embedding) *A mapping $f : X \to Y$ of a metric space (X, d_X) to a metric space (Y, d_Y) is an* embedding with distortion α *if there exists a constant $r > 0$ such that for every $x_1, x_2 \in X$,*

$$r \cdot d_X(x_1, x_2) \leq d_Y(f(x_1), f(x_2)) \leq \alpha r \cdot d_X(x_1, x_2)$$

1.1.4 Concentration Inequalities

The **Chernoff Bound** [27] gives exponentially decreasing bounds on tail distributions of sums of independent random variables. There are many forms (inequalities) of Chernoff bounds, but in this thesis, we will use the following two simplified versions.

Theorem 1 *[27] Let $Y = \sum_{i=1}^{n} Y_i$ where $Y_i = 1$ with probability p_i and 0 with probability $1 - p_i$, and all Y_i's are independent. With $\mu = \mathbb{E}[Y]$, $\mathbb{P}[Y > 2\mu] \leq e^{-\mu/3}$ and $\mathbb{P}\big[Y < \frac{\mu}{2}\big] \leq e^{-\mu/8}$.*

1.2 Problems Considered

Capacitated Vehicle Routing Problem (CVRP) : The classic Capacitated Vehicle Routing Problem was introduced by Dantzig and Ramser in 1959 [16]. In CVRP, we are given as input the locations of n customers and a depot r, along with a vehicle of capacity Q, and wish to compute a minimum length collection of tours, each starting and ending at the depot and visiting at most Q customers, and whose union covers all the customers. We can also view the problem in the following way: given a graph $G = (V, E)$ with a depot $r \in V$, edge cost (or weight) $w : E \to \mathbb{Z}^{\geq 0}$, a demand function $d : V \to \mathbb{Z}^{\geq 0}$ and $Q > 0$, the objective of CVRP is to find a set of tours of minimum total cost each of which includes r such that the union of the tours covers the demand at every client and every tour covers at most Q demand. The following are the three common variants of CVRP that have been studied:

- **Unsplittable CVRP**: In this variant of CVRP, the entire demand of a client must be delivered by a single tour.
- **Unit Demand CVRP**: This variant of CVRP is a special case of unsplittable CVRP where every client has demand one and the demand of a client must be delivered by a single tour.
- **Splittable CVRP**: In this variant of CVRP, the demand of a client could be delivered using multiple tours.

CVRP has also been referred to as the k-tours problem [4], [5].

1.3 Related Work

For general metrics, Haimovich et al. [20] gave a $(1+(1-1/Q)\alpha)$-approximation for splittable CVRP, and a $(2+(1-2/Q)\alpha)$-approximation for unsplittable CVRP where α is the approximation ratio of TSP. Recently, Blauth et al. [11] showed that given a TSP approximation α, there is an $\epsilon > 0$ such that there is an $(\alpha + 2 \cdot (1-\epsilon))$-approximation algorithm for CVRP. For $\alpha = 3/2$, they showed $\epsilon > 1/3000$. They also showed a $(\alpha + 1 - \epsilon)$-approximation algorithm for unit demand CVRP and splittable CVRP. For $\alpha = 3/2$, they showed $\epsilon > 1/3000$.

For the case of trees, Labbé et al. [25] showed splittable CVRP is NP-hard and Golden et al. [19] showed unsplittable CVRP is APX-hard and hard to approximate better than 1.5. For splittable CVRP in trees, Hamaguchi et al. [21] defined the following lower bound for the cost of the optimal solution. For an edge e, let $d(e)$ denote the total demand across all nodes in the subtree rooted at e and let $w(e)$ denote the cost or weight of edge e. Any feasible solution covering demands of nodes in the subtree rooted at e requires at least $\left\lceil \frac{d(e)}{Q} \right\rceil$ tours to pass through e.

$$\mathrm{LB}(e) = 2 \cdot w(e) \cdot \left\lceil \frac{d(e)}{Q} \right\rceil$$

$$\mathrm{LB} = \sum_{e \in E} \mathrm{LB}(e) = 2 \sum_{e \in E} w(e) \cdot \left\lceil \frac{d(e)}{Q} \right\rceil$$

Let OPT be the cost of an optimal solution to the instance. Clearly, $\mathrm{LB} \leq \mathrm{OPT}$. A 1.5 approximation with respect to the lower bound was shown by [21]. Asano et al. [5] improved the approximation to $(\sqrt{41}-1)/4$ with respect to the lower bound and also showed the existence of instances whose optimal cost is exactly 4/3 times the lower bound. Becker and Paul [6] gave a 4/3-approximation with respect to the lower bound, making it tight with respect to the lower bound. Becker and Paul [10] showed a $(1, 1+\epsilon)$-bicriteria

polynomial-time approximation scheme for splittable CVRP in trees i.e. an algorithm that has cost no more than an optimal solution, but each tour covers at most $(1+\epsilon)Q$ nodes. At the time of writing this, these are the best known results for CVRP on trees.

The results above hold for arbitrary Q. For fixed capacity $Q \geq 3$, CVRP is APX-hard in general metrics. When Q is fixed, CVRP is polynomial-time solvable in trees. There exists a polynomial-time approximation scheme (PTAS) for CVRP in the Euclidean plane ($\mathbb{R}^2$) when Q is fixed as shown by Khachay et al. [22]. For arbitrary Q, Das and Mathieu [17] gave a quasi-polynomial-time approximation scheme (QPTAS) for CVRP in the Euclidean plane ($\mathbb{R}^2$) with running time $n^{\log^{O(1/\epsilon)} n}$. A PTAS when Q is $O(\log n / \log\log n)$ or Q is $\Omega(n)$ was shown by Asano et al. [5]. A PTAS for Euclidean plane ($\mathbb{R}^2$) for all moderately large values of $Q \leq 2^{\log^{\delta} n}$, where $\delta = \delta(\epsilon)$ was shown by Adamaszek et al [3], building on the work of Das and Mathieu [17], and using it as a subroutine. For high dimensional Euclidean spaces $\mathbb{R}^d$, Khachay et al. [22] showed a PTAS when Q is $O(\log^{1/d} n)$. For graphs of bounded doubling dimension, Khachay et al. [23] gave a QPTAS when the number of tours is polylog(n) and Khachay et al. [24] gave a QPTAS when Q is polylog(n).

There is a rich literature for CVRP when Q is fixed. A PTAS for planar graphs was shown by Becker et al. [9]. A QPTAS for planar and bounded-genus graphs was shown by Becker et al. [7]. A PTAS for graphs of bounded highway dimension and an exact algorithm for graphs with treewidth with running time $O(n^{\mathrm{tw}Q})$ was shown by Becker et al [8]. A PTAS is an efficient PTAS (an EPTAS) if its running time is bounded by a polynomial n^c whose degree c does not depend on ϵ. Cohen-Addad et al. [13] showed an efficient PTAS for the graphs of bounded-treewidth, an efficient PTAS for bounded highway dimension, an efficient PTAS for bounded genus metrics and a QPTAS for minor-free metrics.

1.4 New Results

A major open question has been to design approximation schemes for other classes of graphs. Even for the case of tree metrics, the best approximation algorithms are constant factor. In this thesis, we make progress by presenting approximation schemes for several classes of graphs and improving previously known results. The main contributions of this thesis are the following.

The Capacitated Vehicle Routing Problem in Trees. In Chapter 2, we prove a theorem that characterizes structural properties of a near-optimal set of tours and use dynamic programming to compute such a solution in quasi-polynomial time to show the following:

Theorem 2 *For any $\epsilon > 0$, there is an algorithm that, for any instance of the Unit Demand CVRP in trees (T, Q) where T is a tree with n vertices, outputs a $(1+\epsilon)$-approximate solution in time $n^{O(\log^4 n/\epsilon^3)}$.*

We will then show how we can extend our dynamic program to also compute a near-optimal solution for Splittable CVRP in trees when $Q = n^{O(\log^c n)}$ to get the following Corollary.

Corollary 1 *For any $\epsilon > 0$, there is an algorithm that, for any instance of the splittable capacitated vehicle routing problem on trees (T, Q) where T is a tree with n vertices, output a $(1+\epsilon)$-approximate solution in time $n^{O(\log^{2c+4} n)}$ when $Q = n^{O(\log^c n)}$.*

The Capacitated Vehicle Routing Problem in Graphs of Bounded Treewidth. In Chapter 3, we will extend our structure theorem for the case of trees to graphs having bounded treewidth. We will compute a near-optimum solution in quasi-polynomial time by showing the following:

Theorem 3 *For any $\epsilon > 0$, there is an algorithm that, for any instance of Unit Demand CVRP on bounded treewidth graphs (G, Q) where G is a graph with n vertices and treewidth k, outputs a solution of expected cost $(1+\epsilon)$ times the optimal solution in time $n^{O(k^2 \log^3 n/\epsilon^2)}$*

We will then show how we can extend our dynamic program to also compute a near-optimal solution for Splittable CVRP in graphs of bounded treewdith when $Q = n^{O(\log^c n)}$ to get the following corollary.

Corollary 2 *Let $\epsilon > 0$, there is an algorithm that, for any instance of the splittable capacitated vehicle routing problem on bounded treewidth graphs (G, Q) where G is a graph with n vertices, treewidth k and when $Q = n^{O(\log^c n)}$, outputs a solution of expected cost $(1+\epsilon)$ times the optimal solution in time $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$*

The Capacitated Vehicle Routing Problem in Graphs of Bounded Doubling Dimension. In Chapter 4, we will use a Lemma of [28] to embed a graph of doubling dimension D into a graph having treewidth at most $2^{O(D)} \left(\frac{8D \log \Delta}{\epsilon}\right)^D$ with expected distortion $1+\epsilon$ where Δ is the aspect ratio. We will then use our algorithm for graphs of bounded treewidth from Chapter 3 as a black box to derive a quasi-polynomial time approximation scheme for Splittable CVRP for graphs of bounded doubling dimension.

Theorem 4 *For any $\epsilon > 0$ and $D > 0$, there is a an algorithm that, given an instance of Splittable CVRP with capacity $Q = n^{O(\log^c n)}$ and if the graph has doubling dimension D with an optimal solution having cost* OPT*, finds a solution whose cost is at most $(1+\epsilon)$*OPT *in time $n^{O(D^D \log^{2c+D+3} n/\epsilon^{D+2})}$.*

As an immediate corollary, this implies an approximation scheme for CVRP on Euclidean metrics on $\mathbb{R}^2$ in time $n^{O(\log^6 n/\epsilon^5)}$ which improves on the run time of $n^{\log^{O(1/\epsilon)} n}$ of QPTAS of [17].

The Capacitated Vehicle Routing Problem in Graphs of Bounded Highway Dimension. In Chapter 4, we will use a Lemma of [18] to embed a graph of highway dimension D and violation λ into a graph having treewidth at most $(\log \Delta)^{O\left(\log^2(\frac{D}{\epsilon\lambda})/\lambda\right)}$ with expected distortion $1+\epsilon$ where Δ is the aspect ratio. We will then use our algorithm for graphs of bounded treewidth from Chapter 3 to derive a quasi-polynomial time approximation scheme for Splittable CVRP for graphs of bounded highway dimension.

Theorem 5 *For any $\epsilon > 0, \lambda > 0$ and $D > 0$, there is a an algorithm that, given an instance of the splittable CVRP with capacity $Q = n^{\log^c n}$ and if the graph has highway dimension D with violation λ with an optimla solution of cost* OPT*, finds a solution whose cost is at most $(1+\epsilon)$*OPT *in time $n^{O(\log^{2c+3+\log^2(\frac{D}{\epsilon\lambda})\cdot\frac{1}{\lambda}} n/\epsilon^2)}$.*

1.5 General Assumptions

An instance $\mathcal{I}$ to CVRP is a graph $G = (V, E)$, where $w(e)$ is the cost or weight of edge $e \in E$ and Q is the capacity of the vehicle. We will use weight and cost interchangeably. Each tour $\mathcal{T}$ is a walk over some nodes of G. We say $\mathcal{T}$ "covers" node v if it serves the demand at node v. It is easier to think of the demand of each node v as being a token on v that must be picked up by a tour. We will use tokens and demand interchangeably. In Unit Demand CVRP, we assume each node v has a single token and for Splittable CVRP, we assume each node v can have multiple tokens and the total number of tokens a tour can pick is most Q (possibly from the same or different locations). Note that each tour might visit vertices without picking any token there. The goal is to find a collection of tours of minimum total cost such that each token is picked up (or say covered) by some tour. We use $\mathrm{OPT}(G)$ or simply OPT to refer to an optimum solution of G, and OPT to denote the value of it.

1.5.1 Total number of tokens and tours

Given an instance for Splittable CVRP with n nodes and capacity Q, it is possible that the demand $d(v) > Q$ for some node v. In order to achieve a running time of quasi-polynomial (as we will see later), it is important that the instance we solve has the property that $d(v) < \mathrm{poly}(n) \cdot \mathrm{poly}(Q)$. From the work of Adamaszek et al [3], we will show how we can assume that the demand at each node v, $d(v)$, is integral and satisfies $1 \leq d(v) < nQ$. Adamaszek et al [3] defined a *trivial* tour to be a tour which picks up tokens from a single node in T and a tour is *non-trivial* if the tour picks up tokens from at least two nodes in T. They defined a *cycle* of tours to be a set of tours $t_1, \ldots, t_m (m \geq 2)$

and a set of nodes $l_1, l_2, \ldots, l_m, l_{m+1} = l_1$ such that each tour t_i covers locations l_i and l_{i+1} and the origin is not considered as a node in $l_1, \ldots, l_m$. They showed in Lemma 1 of [3] that there is an optimal solution in which there are no cycles of tours. Since there are no 2-cycles, there are no two tours which cover the same pair of nodes. Since there are n nodes and we cannot have a cycle of non-trivial tours, there is an optimal solution such that there are at most $n-1$ non-trivial tours. If there are more non-trivial tours, then there is a cycle of tours, as argued in [3]. So putting aside trivial tours (each picking up Q tokens at a node), we can assume we have a total of at most nQ tokens and in particular, each node has at most this many tokens. Without loss of generality, we assume we have removed all trivial tours and so there is a total of at most nQ demand.

We can also assume there is at most one tour in OPT covering at most $Q/2$ demand. If there are at least two tours $\mathcal{T}_1$ and $\mathcal{T}_2$ covering less than $Q/2$ demand, they can be merged into a single tour called $\mathcal{T}$ and $\mathcal{T}$ forms a capacity respecting tour since it covers at most Q demand at no additional cost. Since the total demand is at most nQ, the total number of tours in the optimal solution is at most $nQ/(Q/2) = 2n$.

1.5.2 Poly(n) bounded edge weights

Fix an optimal solution OPT. Without loss of generality, we can assume that each tour in OPT traverses an edge e at most once in each direction. Suppose there is a solution where some tour $\mathcal{T}$ traverses an edge multiple times, we can short-cut the solution at no additional cost such that $\mathcal{T}$ traverses the edge only once in each direction. For an edge e, let $f(e)$ denote the number of tours traversing through edge e in either direction in OPT; so $\textsc{opt} = \sum_e w(e) \cdot f(e)$.

Now, we scale edge weights to be polynomially bounded at a small loss. Suppose we have guessed the largest edge weight that belong to OPT (by enumerating over all possible such guesses) and have removed any edge with weight

larger. Let $W = \max_{e \in E} w(e)$ be the largest (guessed) edge in OPT. We will create a new instance G' by rounding up the cost of every edge to be at least $\frac{\epsilon W}{4n^3}$. Let $w'(e)$ be the extra cost added to each edge when we round up its cost. Recall $\textsc{opt} = \sum_{e \in E} w(e) \cdot f(e)$. We will use the fact that $f(e) \leq 4n$ since there are at most $2n$ tours in OPT. Note that $|E| < n^2$ and $W \leq \textsc{opt}$. Let $\textsc{opt}'$ denote the cost of the optimal solution to the new instance G'.

$$\begin{aligned}
\textsc{opt}' &\leq \sum_{e \in E} (w(e) + w'(e)) \cdot f(e) \\
&\leq \sum_{e \in E} w(e) \cdot f(e) + \sum_{e \in E} \frac{\epsilon W}{4n^3} f(e) \\
&\leq \textsc{opt} + \sum_{e \in E} \frac{\epsilon W}{4n^3} \cdot 4n \\
&< \textsc{opt} + \frac{\epsilon W}{4n^3} \cdot 4n \cdot n^2 \\
&= \textsc{opt} + \epsilon W \\
&\leq (1+\epsilon)\textsc{opt}.
\end{aligned}$$

We showed that by rescaling the edge cost, the total cost of the solution increases by at most $\epsilon \cdot \textsc{opt}$. Note that for any edge e,

$$\frac{\epsilon W}{4n^3} \leq w(e) + w'(e) \leq W.$$

The ratio between the maximum edge cost in G' and the minimum edge cost in G' is at most $\frac{4n^3}{\epsilon}$, so we can rescale the weights so the the weight of an edge is between $1/\epsilon$ and $4n^3/\epsilon^2$. However, the weights of the edges in G' might not be integral. We will create a new instance G'' and show we can assume the weights are integral and in the range $[1/\epsilon, 4n^3/\epsilon^2]$ at a small loss of ϵ times the cost of the optimal solution. We will create a new instance G'' where we round-up the weight of the edge to the nearest integer. Notice that since the minimum weight of an edge is $1/\epsilon$, the maximum additional cost of rounding-up the weight of an edge to the nearest integer is at most ϵ fraction of the weight of an edge i.e. $w_{G''}(e) \leq (1+\epsilon) w_{G'}(e)$. For any tour $\mathcal{T}$, let $\text{cost}_{\mathcal{T}}(G'')$ denote the cost of $\mathcal{T}$ in instance G''. Since $w_{G''}(e) \leq (1+\epsilon) w_{G'}(e)$, we have that

$$\text{cost}_{\mathcal{T}}(G'') = \sum_{e \in \mathcal{T}} w_{G''}(e) \leq (1+\epsilon) \sum_{e \in \mathcal{T}} w_{G'}(e) = (1+\epsilon)\text{cost}_{\mathcal{T}}(G')$$

Let $\textsc{opt}''$ denote the cost of the optimal solution in instance G''. Let OPT' and OPT'' denote the optimal solution to instances G' and G''. We have that

$$\begin{aligned}\textsc{opt}'' &= \sum_{\mathcal{T}\in\mathrm{OPT}''} \mathrm{cost}_{\mathcal{T}}(G'') \leq \sum_{\mathcal{T}\in\mathrm{OPT}'} \mathrm{cost}_{\mathcal{T}}(G'') \\ &\leq (1+\epsilon) \sum_{\mathcal{T}\in\mathrm{OPT}'} \mathrm{cost}_{\mathcal{T}}(G') \\ &= (1+\epsilon)\textsc{opt}'\end{aligned}$$

We showed that we can find an instance G'' such that edges in G'' are integral in the range $[1/\epsilon, 4n^3/\epsilon^2]$ and has the property that the cost of the set of optimal set of tours in G'' lifted to G would have cost $(1+\epsilon)^2\textsc{opt}$.

Recall we denoted the aspect ratio by $\Delta = \frac{d_{\max}}{d_{\min}}$. Since the edge costs are are integral and between$[1/\epsilon, 4n^3/\epsilon^2]$, $\Delta = \frac{4n^3}{\epsilon}$ and $\log \Delta \leq 12 \log\left(\frac{n}{\epsilon}\right)$.

Chapter 2

QPTAS for CVRP in Trees

We first consider the *Capacitated Vehicle Routing Problem* (CVRP) in trees.

2.1 Problem Overview

Given an instance of CVRP where the graph is a tree, we will assume the depot is located at the root of the tree. For the case of *Unit Demand CVRP*, we will assume the demand at every node is 1 and the entire demand at a node must be delivered by a single vehicle. For the case of *Splittable CVRP*, we will assume each node has a demand d_v satisfying $0 \leq d_v < nQ$. The demand at a node could be delivered using multiple vehicles.

2.1.1 Our Results

We will prove a structure theorem which describes structural properties of a near-optimal solution. We will leverage these structural properties and use dynamic programming to compute a near-optimum solution. We will state Theorem 2 and Corollary 1 for convenience.

Theorem 2 *For any $\epsilon > 0$, there is an algorithm that, for any instance of the Unit Demand CVRP in trees (T, Q) where T is a tree with n vertices, outputs a $(1+\epsilon)$-approximate solution in time $n^{O(\log^4 n/\epsilon^3)}$.*

We will then show how we can extend our dynamic program to also compute a near-optimal solution for Splittable CVRP in trees when $Q = n^{O(\log^c n)}$ to get the following corollary.

Corollary 1 *For any $\epsilon > 0$, there is an algorithm that, for any instance of the splittable capacitated vehicle routing problem on trees (T, Q) where T is a tree with n vertices, output a $(1+\epsilon)$-approximate solution in time $n^{O(\log^{2c+4} n)}$ when $Q = n^{O(\log^c n)}$.*

We will first prove the structure theorem.

2.2 Structure Theorem

Our goal in this section is to show the existence of a near optimum solution (i.e. one with cost $(1+O(\epsilon))\text{OPT}$) with certain properties which makes it easy to find one using dynamic programming. More specifically we show we can modify the instance $\mathcal{I}$ to instance $\mathcal{I}'$ on the same tree T where each node has ≥ 1 tokens (so possibly more than 1) and change OPT to a solution OPT' on $\mathcal{I}'$ where cost of OPT' is at most $(1+O(\epsilon))\text{OPT}$. Clearly the tours of OPT' form a capacity respecting solution of $\mathcal{I}$ as well (of no more cost).

A starting point in our structure theorem is to show that given input tree T, for any $\epsilon > 0$, we can build another tree T' of height $O(\log^2 n/\epsilon)$ such that the cost of an optimum solution in T' is within $1+\epsilon$ factor of the optimum solution to T. We can lift a near-optimum solution to T' into a near-optimum solution of T. We will show the following in Subsection 2.5

Theorem 6 *Given a tree T as an instance of CVRP and for any fixed $\epsilon > 0$, one can build a tree T' with height $\delta \log^2 n/\epsilon$, for some fixed $\delta > 0$, such that $\text{OPT}(T') \leq \text{OPT}(T) \leq (1+\epsilon)\text{OPT}(T')$.*

So for the rest of this section we assume our input tree has height $O(\log^2 n/\epsilon)$ at a loss of (yet another) $1+\epsilon$ in approximation ratio.

2.2.1 Overview of the ideas

Before diving right into the details of Structure theorem, we will first motivate our core idea by starting from a simpler task of developing a bi-criteria approximation scheme [1]. In our case, a $(1+\epsilon)$-*bi-criteria* approximation scheme

[1]Note that [10] already presents a bicriteria PTAS for CVRP on trees.

is an algorithm that given an instance $\mathcal{I}$ for CVRP with optimal solution OPT, returns a solution of cost at most OPT such that the demand delivered by each tour is at most $(1+\epsilon)Q$.

Let $\mathcal{T}$ be a tour in OPT and v be a node in T. The **coverage** of $\mathcal{T}$ with respect to v is the number of tokens picked by $\mathcal{T}$ in the subtree T_v. Suppose a tour $\mathcal{T}$ visits node v. We refer to the subtour of $\mathcal{T}$ in T_v (subtree rooted at v) as a partial tour.

A Bicriteria QPTAS: For simplicity, assume T is binary (this is not crucial in the design of the DP). A subproblem would be based on a node $v \in T$ and the structure of partial tours going into T_v to pick up tokens in T_v at minimum cost. In other words, if one looks at the sections of tours of an optimum solution that cover tokens of T_v, what are the capacity profiles of those sections? For a vector $\vec{t}$ with Q entries, where $\vec{t}_i$ (for each $1 \leq i \leq Q$) is the number of partial tours going down T_v which pick i tokens (or their capacity for that portion is i), entry $\mathbf{A}[v, \vec{t}]$ would store the minimum cost of covering T_v with (partial) tours whose capacity profile is given by $\vec{t}$. It is not hard to fill this table's entries using a simple recursion based on the entries of children of v. So one can solve the CVRP problem "exactly" in time $O(n^{Q+1})$. We can reduce the time complexity by storing "approximate" sizes of the partial tours for each T_v. So let us "round" the capacities of the tours into $O(\log Q/\epsilon)$ buckets, where bucket i represents capacities that are in $[(1+\epsilon)^{i-1}, (1+\epsilon)^i)$. More precisely, consider threshold-sizes $S = \{\sigma_1, \ldots, \sigma_\tau\}$ where: for $1 \leq i \leq 1/\epsilon$, $\sigma_i = i$, and for each value $i > 1/\epsilon$: $\sigma_i = \sigma_{i-1}(1+\epsilon)$ and $\sigma_\tau = Q$. Note that $|S| = O(\log Q/\epsilon) = O(\log n/\epsilon)$. Suppose we allow each tour to pick up to $(1+\epsilon)Q$ tokens. If it was the case that each partial tour for T_v (i.e. part of a tour that enters/exits T_v) has a capacity that is also threshold-size (this may not be true!) then the DP table entries would be based on vectors $\vec{t}$ of size $O(\log n/\epsilon)$, and the run time would be quasi-polynomial. One has to note that for each subproblem of the optimum at a node v with children u, w, even if the tour sizes going down T_v were of threshold-sizes, the partial tours at T_u

and T_w do not necessarily satisfy this property.

To extend this to a proper bicriteria $(1+\epsilon)$-approximation we can define the thresholds based on powers of $1+\epsilon'$ where $\epsilon' = \frac{\epsilon^2}{\log^2 n}$ instead: let $S = \{\sigma_1, \ldots, \sigma_\tau\}$ where $\sigma_i = i$ for $1 \leq i \leq 1/\epsilon'$, and for $i > 1/\epsilon'$ we have $\sigma_i = \sigma_{i-1}(1+\epsilon')$, and $\sigma_\tau = Q$. So now $|S| = O(\log^2 n \cdot \log Q/\epsilon) = O(\log^3 n/\epsilon^2)$ when $Q = \text{poly}(n)$. For each vector $\vec{t}$ of size τ, where $0 \leq t_i \leq n$ is the number of partial tours with coverage/capacity σ_i, let $A[v, \vec{t}]$ store the minimum cost of a collection of (partial) tours covering all the tokens in T_v whose capacity profile is $\vec{t}$, i.e. the number of tours of size in $[\sigma_i, \sigma_{i+1})$ is $\vec{t}_i$. To compute the solution for $A[v, \vec{t}]$, given all the solutions for its two children u, w we can do the following: consider two partial solutions, $A[u, \vec{t}_u]$ and $A[w, \vec{t}_w]$. One can combine some partial tours of $A[u, \vec{t}_u]$ with some partial tours of $A[w, \vec{t}_w]$, i.e. if $\mathcal{T}_u$ is a (partial) tour of class i for T_u and $\mathcal{T}_w$ is a partial tour of class j for T_w then either these two tours are in fact part of the same tour for T_v, or not. In the former case, the partial tour for T_v obtained by the combination of the two tours will have cost $w(\mathcal{T}_u) + w(\mathcal{T}_w) + 2w(vu) + 2w(vw)$ and capacity $t_i + t_j$ (or possibly $t_i + t_j + 1$ if this tour is to cover v as well). In the latter case, each of $\mathcal{T}_u$ and $\mathcal{T}_w$ extend (by adding edges vu and vw, respectively) into partial tours for T_v of weights $w(\mathcal{T}_u) + 2w(vu)$ and $w(\mathcal{T}_w) + 2w(vw)$ (respectively) and capacities t_i and t_j (or perhaps $t_i + 1$ or $t_j + 1$ if one of them is to cover v as well). In the former case, since $t_i + t_j$ is not a threshold-size, we can round it (down) to the nearest threshold-size. We say partial solutions for T_v, T_u and T_w are consistent if one can obtain the partial solution for T_v by combining the solutions for T_v and T_w. Given $A[v, \vec{t}]$, we consider all possible subproblems $A[u, \vec{t}_u]$ and $A[w, \vec{t}_w]$ that are consistent and take the minimum cost among all possible ways to combine them to compute $A[v, \vec{t}]$. Note that whenever we combine two solutions, we might be rounding the partial tour sizes down to a threshold-size, so we "under-estimate" the actual tour size by a factor of $1+\epsilon'$ in each subproblem calculation. Since the height of the tree is $h = O(\log^2 n/\epsilon)$, the actual error in the tour sizes computed at the root is at most $(1+\epsilon')^h = (1+O(\epsilon))$, so each tour will have size at most $(1+O(\epsilon))Q$. The time to compute each entry $A[v, \vec{t}]$ can be upper bounded by $n^{O(\log^3 n/\epsilon^2)}$

and since there are $n^{O(\log^3 n/\epsilon^2)}$ subproblems, the total running time of the algorithm will be $n^{O(\log^3 n/\epsilon^2)}$. We can handle the setting where the tree is not binary (i.e. each node v has more than two children) by doing an inner DP, like a knapsack problem over children of v (we skip the details here as we will explain the details for the actual QPTAS instead).

Going from a Bicriteria to a true QPTAS: Our main tool to obtain a true approximation scheme for CVRP in trees is to show the existence of a near-optimum solution where the partial solutions for each T_v have sizes that can be grouped into polyogarithmic many buckets as in the case of bi-criteria solution. Roughly speaking, starting from an optimum solution OPT, we follow a bottom-up scheme and modify OPT by changing the solution at each T_v: at each node v, we change the structure of the tours going down T_v (by adding a few extra tours from the depot) and also adding some extra tokens at v so that the partial tours that visit T_v all have a size from one of polyogarithmic many possible sizes (buckets) while increasing the number and the cost of the tours by a small factor. We do this by duplicating some of the tours that visit T_v while changing parts of them that go down in T_v and adding some extra tokens at v: each tour still picks up at most a total of Q tokens and the size (i.e. the number of tokens picked) for each partial tour in the subtree T_v is one of $O(\log^4 n/\epsilon^2)$ many possible values, while the total cost of the solution is at most $(1 + O(\epsilon))\textsc{opt}$.

Suppose T has height h (where $h = \delta \log^2 n/\epsilon$). Let V_ℓ (for $1 \leq \ell \leq h$) be the set of vertices at level ℓ of the tree where $V_1 = \{r\}$ and for each $\ell \geq 2$, V_ℓ are those vertices whose parent is in level $\ell - 1$. For every tour $\mathcal{T}$ and every level ℓ, the top part of $\mathcal{T}$ w.r.t. ℓ (denoted by $\mathcal{T}_\ell^{top}$), is the part of $\mathcal{T}$ induced by the vertices in $V_1 \cup \ldots \cup V_{\ell-1}$ and the bottom part of $\mathcal{T}$ are the partial tours of $\mathcal{T}$ in the subtrees rooted at a vertex in V_ℓ. Note that if we replace each partial tour of the bottom part of a tour $\mathcal{T}$ with a partial tour of a smaller capacity, the tour remains a capacity respecting tour. Consider a node v (which is at some level ℓ) and suppose we have n_v partial tours covering T_v. Let the n_v tours in increasing order of their coverage be $t_1, \ldots, t_{n_v}$. Let

$|t_i|$ be the coverage of tour t_i (so $|t_i| \leq |t_{i+1}|$). For a value g (to be specified later), we add enough empty tours to the beginning of this list so that the number of tours is a multiple of g. Then, we will put these tours into groups $G_1^v, \ldots, G_g^v$ of equal sizes by placing the i'th n_v/g partial tours into G_i^v. Let $h_i^{v,max}$ ($h_i^{v,min}$) refer to the maximum (minimum) size of the tours in G_i^v. This grouping is similar to the grouping in the asymptotic PTAS for the classic bin-packing problem. Note that $h_i^{v,max} \leq h_{i+1}^{v,min}$.

Consider a mapping f where it maps each partial tour in G_i^v to one in G_{i-1}^v in the same order, i.e. the largest partial tour in G_i^v is mapped to the largest in G_{i-1}^v, the 2nd largest to the 2nd largest and so on, for $i > 1$ (suppose $f(.)$ maps all the tours of G_1^v to empty tours). Now suppose we modify OPT to OPT$'$ in the following way: for each tour $\mathcal{T}$ that has a partial tour $t \in G_i^v$, replace the bottom part of $\mathcal{T}$ at v from t to $f(t)$ (which is in G_{i-1}^v). Note that by this change, the size of any tour like $\mathcal{T}$ can only decrease. Also, if instead of $f(t)$ we had replaced t with a partial tour of size $h_{i-1}^{v,max}$, it would still form a capacity respecting solution with the rest of $\mathcal{T}$, because $h_{i-1}^{v,max} \leq h_i^{v,min} \leq |t|$. The only problem is that those tokens in T_v that were picked by the partial tours in G_g^v are not covered by any tours; we call these *orphant* tokens. For now, assume that we add a few extra tours to OPT at low cost such that they cover all the orphant tokens of T_v. If we have done this change for all vertices $v \in V_\ell$, then for every tour like $\mathcal{T}$, the partial tours of $\mathcal{T}$ going down each T_v (for $v \in V_\ell$) are replaced with partial tours from a group one index smaller. This means that, after these changes, for each tour $\mathcal{T}$ and its (new) partial tour $t \in G_i^v$, if we add $h_i^{v,max} - |t|$ extra tokens at v to be picked up by t then each partial tour has size exactly the same as the maximum size of its group without violating the capacities. This helps us store a compact "sketch" for partial solutions at each node v with the property that the partial solution can be extended to a near optimum one.

How to handle the case of orphant tokens (those picked by the tours in the the last groups G_g^v before the swap)? We will show that if n_v is sufficiently large (at least polylogarithmic) then if we sample a small fraction of the tours of the optimum at random and add two copies of them (as extra tours), they

can be used to cover the orphant tokens. So overall, we show how one can modify OPT by adding some extra tours to it at a cost of at most $\epsilon \cdot \text{OPT}$ such that: each node v has ≥ 1 tokens and the sketch of the partial tours at each node v is compact (only polyogarithmic many possible sizes) while the dropped tokens overall can be covered by the extra tours.

2.2.2 Changing OPT to a near optimum structured solution

We will show how to modify the optimal solution OPT to a near-optimum solution OPT$'$ for a new instance $\mathcal{I}'$ which has ≥ 1 token at each node with certain properties. We start from $\ell = h$ and let $\text{OPT}' = \text{OPT}_\ell = \text{OPT}$ and for decreasing values of ℓ, we will show how to modify $\text{OPT}_{\ell+1}$ to obtain OPT_ℓ. To obtain OPT_ℓ from $\text{OPT}_{\ell+1}$ we keep the partial tours at levels $\geq \ell$ the same as $\text{OPT}_{\ell+1}$ but we change the top parts of the tours and how the top parts can be matched to the partial tours at level ℓ so that together they form capacity respecting solutions (tours of capacity at most Q) at low cost.

First, we assume that OPT has at least $d \log n$ many tours for some sufficiently large d. Otherwise, if there are at most $D = d \log n$ many tours in OPT we can do a simple DP to compute OPT: for each node v, we have a sub problem $A[v, T_1^v, \ldots, T_D^v]$ which stores the minimum cost solution if T_i^v is the number of vertices the i'th tour is covering in the subtree T_v. It is easy to fill this table in time $O(n^D)$ having computed the solutions for its children.

Definition 3 *Let* ***threshold values*** *be* $\{\sigma_1, \ldots, \sigma_\tau\}$ *where* $\sigma_i = i$ *for* $1 \leq i \leq \lceil 1/\epsilon \rceil$*, and for* $i > \lceil 1/\epsilon \rceil$ *we have* $\sigma_i = \lceil \sigma_{i-1}(1+\epsilon) \rceil$*, and* $\sigma_\tau = Q$*. So* $\tau = O(\log Q/\epsilon)$*.*

We consider the vertices of T level by level, starting from nodes in level $V_{\ell=h-1}$ and going up, modifying the solution $\text{OPT}_{\ell+1}$ to obtain OPT_ℓ.

Definition 4 *For a node* v*, the* i*-th bucket,* b_i*, contains the number of tours of* OPT_ℓ *having coverage between* $[\sigma_i, \sigma_{i+1})$ *tokens in* T_v *where* σ_i *is the* i*-th threshold value. We will denote a node and bucket by a pair* (v, b_i)*. Let* $n_{v,i}$ *be the number of tours in bucket* b_i *of* v*.*

Definition 5 *A bucket b is **small** if the number of tours in b is at most $\alpha \log^3 n/\epsilon^2$ and is **big** otherwise, for a constant $\alpha \geq \max\{1, 12\delta\}$.*

Note that for every node v and bucket b_i and for any two partial tours in b_i, the ratio of their size (coverage) is at most $(1+\epsilon)$. We will use this fact crucially later on. While giving the high level idea earlier in this section, we mentioned that we can cover the orphant tokens at low cost by using a few extra tours at low cost. For this to work, we need to assume that the ratio of the maximum size tour to the minimum size tour in all groups $G_1^v, \ldots, G_g^v$ is at most $(1+\epsilon)$. To have this property, we need to do the grouping described for each vertex-bucket pair (v, b_i) that is big.

For each $v \in V_\ell$, let (v, b_i) be a vertex-bucket pair. If b_i is a small bucket, we do not modify the partial tours in it. If b_i is a big bucket, we create groups $G_{i,1}^v, \ldots, G_{i,g}^v$ of equal sizes (by adding null/empty tours if needed to $G_{i,1}^v$ to have equal size groups), for $g = (2\delta \log n)/\epsilon^2$; so $|G_{i,j}^v| = \lceil n_{v,i}/g \rceil$. We also consider a mapping f (as before) which maps (in the same order) the tours $t \in G_{i,j}^v$ to the tours in $G_{i,j-1}^v$ for all $1 < j \leq g$. We assume the mapping maps tours of $G_{i,1}^v$ to empty tours. Let the size of the smallest (largest) partial tour in $G_{i,j}^v$ be $h_{i,j}^{v,min}$ ($h_{i,j}^{v,max}$). Note that $h_{i,j-1}^{v,max} \leq h_{i,j}^{v,min}$. Consider the set $\mathbf{T}_\ell$ of all the tours $\mathcal{T}$ in OPT_ℓ that visit a vertex in one of the lower levels $V_{\geq \ell}$. Consider an arbitrary such tour $\mathcal{T}$ that has a partial tour t in a big vertex/bucket pair (v, b_i), suppose t belongs to group $G_{i,j}^v$. We replace t with $f(t)$ in $\mathcal{T}$. Note that for $\mathcal{T}$, the partial tour at T_v now has a size between $h_{i,j-1}^{v,min}$ and $h_{i,j-1}^{v,max}$. Now, add some extra tokens at v to be picked up by $\mathcal{T}$ so that the size of the partial tour of $\mathcal{T}$ at T_v is exactly $h_{i,j-1}^{v,max}$; note that since $h_{i,j-1}^{v,max} \leq |t|$, the new partial tour at v can pick up the extra tokens without violating the capacity of $\mathcal{T}$. If we make this change for all tours $\mathcal{T} \in \mathbf{T}_\ell$, each partial tour of them at level ℓ that was in a group $j < g$ of a big vertex/bucket pair (v, i) is replaced with a smaller partial tour from group $j-1$ of the same big vertex/bucket pair; after adding extra tokens at v (if needed) the size is the maximum size from group $j-1$. All other partial tours (from small vertex/bucket pairs) remain unchanged. Also, the total cost of the tours has not increased (in fact some

now have partial tours that are empty). However, the tokens that were picked by partial tours from $G^v_{i,g}$ for a big vertex/bucket pair (v, b_i) are now orphant. We describe how to cover them with some new tours.

One important observation is that when we make these changes, for any partial tours at vertices at lower levels ($V_{>\ell}$) their size remains the same. It is only the tour sizes going down a vertex at level ℓ that we are adjusting (by adding extra tokens). All other lower level partial tours remain unchanged (only their top parts may get swapped). This property holds inductively as we go up the tree and ensure that the lower level partial tours have one of polylogarithmic many sizes. More precisely, as we go up levels to compute OPT_ℓ, for any vertex $v' \in V_{\ell'}$ (where $\ell' > \ell$) and any partial tour $\mathcal{T}'$ visiting $T_{v'}$, either $|\mathcal{T}'|$ belongs to a small vertex bucket pair $(v', b_{i'})$ (and so has one of $O(\log^3 n/\epsilon)$ many possible values) or if it belongs to a big vertex bucket pair $(v', b_{i'})$ then its size is equal to $h^{v',max}_{i',j'}$ for some group j' and hence one of $O((\log Q \log n)/\epsilon^2)$ possible values.

To handle (cover) orphant nodes, we are going to (randomly) select a subset of tours of OPT as "extra tours" and add them to OPT' and modify them such that they cover all the tokens that are now orphant (i.e. those that were covered by partial tours of $G^v_{i,g}$ for all big vertex/bucket pairs at level ℓ).

Suppose we select each tour $\mathcal{T}$ of OPT with probability ϵ. We make two copies of the extra tour and we designate both extra copies to one of the levels V_ℓ that it visits with equal probability. We call these the extra tours.

Lemma 1 *The cost of extra tours selected is at most* $4\epsilon \cdot \text{OPT}$ *w.h.p.*

Proof. Recall that $f(e)$ denotes the number of tours passing through e in OPT. The contribution of edge e to the optimal solution is $2 \cdot w(e) \cdot f(e)$ and we can write $\text{OPT} = \sum_{e \in E} 2 \cdot w(e) \cdot f(e)$. Let e be the parent edge of a node in $v \in V_\ell$. Suppose an extra tour is designated to level ℓ, we will only use it to cover orphant tokens from big buckets from nodes in V_ℓ. A node v would use an extra tour to cover orphant tokens only if one of v's buckets is a big bucket. From now on, we will assume the extra tours only pass through an edge e if $f(e) \geq \alpha \log^3 n/\epsilon^2$ (we can shortcut it otherwise).

For an edge e, let $f'(e)$ denote the number of sampled tours passing through e and since we use two copies of each sampled tour, $2f'(e)$ is the number of extra tours passing through e in OPT$'$. We can write $\textsc{opt}' = \sum_{e \in E} 2 \cdot w(e) \cdot (f(e) + 2f'(e))$ and the cost of extra tours is $\sum_{e \in E} 2 \cdot w(e) \cdot 2f'(e)$. While modifying OPT to OPT$'$, each tour in the optimal solution is sampled with probability ϵ. Let e be an edge with $f(e)$ tours $\mathcal{T}_{e,1}, \ldots, \mathcal{T}_{e,f(e)}$ passing through it. Let $Y_{e,i}$ be a random variable which is 1 if tour $\mathcal{T}_{e,i}$ is sampled and 0 otherwise.

$$\mathbb{E}[Y_{e,i}] = \mathbb{P}[\mathcal{T}_{e,i} \text{ is sampled}] = \epsilon.$$

Let $f'(e) = Y_e = \sum_{i=1}^{f(e)} Y_{e,i}$. By linearity of expectations, we have

$$\mathbb{E}[f'(e)] = \mathbb{E}[Y_e] = \sum_{i=1}^{f(e)} \mathbb{E}[Y_{e,i}] = \sum_{i=1}^{f(e)} \epsilon = \epsilon \cdot f(e).$$

Our goal is to show $\mathbb{P}[Y_e > 2\mathbb{E}[Y_e]]$ is very low. Using Chernoff bound with $\mu = \mathbb{E}[Y_e] = \epsilon \cdot f(e) \geq \alpha \log^3 n/\epsilon \geq 6 \log n$.

$$\mathbb{P}[Y_e > 2\mathbb{E}[Y_e]] \leq e^{-(2 \log n)} = \frac{1}{n^2}$$

The above concentration bound holds for a single edge e. Using the union bound, we can show this hold with high probability over all edges,

$$\sum_{e \in E} \mathbb{P}[Y_e > 2\mathbb{E}[Y_e]] \leq \frac{1}{n}.$$

We showed $f'(e) \leq 2\epsilon \cdot f(e)$ with high probability. Hence, with high probability, the cost of the extra tours is at most

$$\sum_{e \in E} 2 \cdot w(e) \cdot 2f'(e) \leq \sum_{e \in E} 2 \cdot w(e) \cdot 4\epsilon \cdot f(e) = 4\epsilon \sum_{e \in E} 2 \cdot w(e) \cdot f(e) = 4\epsilon \cdot \textsc{opt}.$$

■

Therefore, we can assume that the cost of all the extra tours added is at most $4\epsilon \cdot \textsc{opt}$. Let X_ℓ be the set of extra tours designated to level ℓ. We assume we add X_ℓ when we are building OPT_ℓ (it is only for the sake of analysis). For each $v \in V_\ell$ and vertex/bucket pair (v, b_i), let $X_{v,i}$ be those in X_ℓ whose partial tour in T_v has a size in bucket b_i. Each extra tour in X_ℓ will not be picking

any of the tokens in levels $V_{<\ell}$ (as they will be covered by the tours already in OPT_ℓ); they are used to cover the orphant tokens created by partial tours of $G^v_{i,g}$ for each big vertex/bucket pair (v, b_i) with $v \in V_\ell$; as described below.

Lemma 2 *For each level V_ℓ, each vertex $v \in V_\ell$ and big vertex/bucket pair (v, b_i), w.h.p. $|X_{v,i}| \geq \frac{\epsilon^2}{\delta \log^2 n} \cdot n_{v,i}$.*

Proof. Suppose (v, b_i) is a big vertex/bucket pair at some level V_ℓ. Let $p_1, \ldots, p_{n_{v,i}}$ be the partial tours in vertex/bucket pair (v, b_i). Let the tour in OPT corresponding to p_i be $\mathcal{T}$. Two copies of tour p_i are assigned to b_i if both of the following events are true:

- Let A_i be the event where tour $\mathcal{T}$ is sampled as an extra tour. Since each tour is sampled with probability ϵ, we have $\mathbb{P}[A_i] = \epsilon$.
- Let B_i be the event where tour $\mathcal{T}$ is assigned to level ℓ. There are $h = \delta \log^2 n/\epsilon$ many levels and since $\mathcal{T}$ (if sampled) is assigned to any one of its levels, $\mathbb{P}[B_i] \geq 1/h \geq \epsilon/(\delta \log^2 n)$.

Let Y_i be a random variable which is 1 if p_i is an extra tour in (v, b_i) and 0 otherwise.

$$\mathbb{E}[Y_i] = \mathbb{P}[Y_i = 1] = \mathbb{P}[A_i \wedge B_i] = \mathbb{P}[A_i] \cdot \mathbb{P}[B_i] \geq \epsilon^2/(\delta \log^2 n).$$

Let $Y_{v,i} = \sum_{i=1}^{n_{v,i}} Y_i$ be the random variable keeping track of the number of sampled tours in (v, b_i). The number of extra tours, $|X_{v,i}| = 2Y_{v,i}$ since we add two copies of a sampled tour to $X_{v,i}$. By linearity of expectation, we have

$$\mathbb{E}[|X_{v,i}|] = 2\mathbb{E}[Y_{v,i}] = 2\sum_{i=1}^{n_{v,i}} \mathbb{E}[Y_i] \geq \frac{2\epsilon^2}{\delta \log^2 n} \cdot n_{v,i}.$$

We want to show that $|X_{v,i}| \geq \frac{\mathbb{E}[|X_{v,i}|]}{2} \geq \frac{\epsilon^2}{\delta \log^2 n} \cdot n_{v,i}$ with high probability over all vertex-bucket pairs.

Using Chernoff Bound with $\mu = \mathbb{E}[|X_{v,i}|] \geq \frac{2\epsilon^2}{\delta \log^2 n} \cdot n_{v,i} \geq 24 \log n$ since $n_{v,i} \geq \alpha \log^3 n/\epsilon^2$ and $\alpha \geq 12\delta$.

$$\mathbb{P}\left[|X_{v,i}| < \frac{\mathbb{E}[|X_{v,i}|]}{2}\right] \leq e^{-(3 \log n)} = \frac{1}{n^3}$$

Note that the above equation only shows the concentration bound for a single vertex/bucket pair. There are n nodes and each node has up to $\tau = \log n/\epsilon$ buckets, so the total number of vertex/bucket pairs is at most $n \log n/\epsilon$. Suppose we do a union bound over all buckets, we get

$$\sum_{\text{all } (v,b_i) \text{ pairs}} \mathbb{P}\left[|X_{v,i}| < \frac{\mathbb{E}[|X_{v,i}|]}{2}\right] \leq \frac{1}{n}.$$

We showed that for each vertex/bucket pair v, b_i, $|X_{v,i}| \geq \frac{\epsilon^2}{\delta \log^2 n} n_{v,i}$ holds with high probability. ■

Lemma 3 *Consider all $v \in V_\ell$, big vertex/bucket pairs (v, b_i) and partial tours in $G^v_{i,g}$. We can modify the tours in $X_{v,i}$ (without increasing the cost) and adding some extra tokens at v (if needed) so that:*

1. *The tokens picked up by partial tours in $G^v_{i,g}$ are covered by some tour in $X_{v,i}$, and*

2. *The new partial tours that pick up the orphant tokens in $G^v_{i,g}$ have size exactly $h^{v,max}_{i,g}$ and all tours still have size at most Q.*

3. *For each (new) partial tour of $X_{v,i}$ and every level $\ell' > \ell$, the size of partial tours of $X_{v,i}$ at a vertex at level ℓ' is also one of $O(\log Q \log^3 n/\epsilon^3)$ many sizes.*

Proof. Our goal is to use the extra tours in $X_{v,i}$ to cover tokens picked up by partial tours of $G^v_{i,g}$ and we want each extra tour in $X_{v,i}$ to cover exactly $h^{v,max}_{i,g}$ tokens. The tours in the last group, $G^v_{i,g}$, cover $\sum_{t \in G^v_{i,g}} |t|$ many tokens. Since we want each tour in $X_{v,i}$ to cover $h^{v,max}_{i,g}$ tokens, we will add $\sum_{t \in G^v_{i,g}} (h^{v,max}_{i,g} - |t|)$ extra tokens at v for each vertex/bucket pair (v, b_i) so that there are $h^{v,max}_{i,g}$ tokens for each partial tour in $G^v_{i,g}$. From now on, we will assume each partial tour in a last group $G^v_{i,g}$ covers $h^{v,max}_{i,g}$ tokens.

We know $|G^v_{i,g}| = n_{v,i}/g = \frac{\epsilon^2}{2\delta \log n} \cdot n_{v,i}$. Using Lemma 2, we know with high probability that $|X_{v,i}| \geq \frac{\epsilon^2}{\delta \log^2 n} \cdot n_{v,i} = 2|G^v_{i,g}|$, so $|X_{v,i}|/|G^v_{i,g}| \geq 2$. Recall OPT$'$ includes tours in OPT plus the extra tours in OPT that were sampled. Let $Y_{v,i}$ denote the number of tours in vertex/bucket pair (v, b_i) that were

sampled, so $|X_{v,i}| = 2|Y_{v,i}|$ since we made two extra copies of each sampled tour and $|Y_{v,i}| \geq |G^v_{i,g}|$ with high probability. We will start by creating a one-to-one mapping $s : G^v_{i,g} \to Y_{v,i}$ which maps each tour in $G^v_{i,g}$ to a sampled tour in $Y_{v,i}$. We know such a one-to-one mapping exists since $|Y_{v,i}| \geq |G^v_{i,g}|$.

Let $\mathcal{T}$ be a sampled tour in $Y_{v,i}$ with two extra copies of it, $\mathcal{T}_1$ and $\mathcal{T}_2$ in $X_{v,i}$. Let the partial tours of $\mathcal{T}$ at the bottom part in V_ℓ be $p_1, \ldots, p_m$. We know $|\mathcal{T}| \geq \sum_{i=1}^{m} |p_i|$. Since s is one-to-one, one partial tour from $r_k \in G^v_{i,g}$ maps to p_j or no tour maps to p_j. If no tour maps to p_j, we consider the load assigned to p_j to be zero. If $s(r_k) = p_j$ where $r_k \in G^v_{i,g}$, since we added extra tokens to make each partial tour $r_k \in G^v_{i,g}$ have $h^{v,max}_{i,g}$ tokens, the load assigned to p_j would be $h^{v,max}_{i,g}$.

Suppose we think of $r_1, \ldots, r_m$ as items and $\mathcal{T}_1$ and $\mathcal{T}_2$ as bins of size Q. We know each r_i fits into a bin of size Q. Recall that for the tour r_j assigned to p_j, we know $|r_j| \leq (1+\epsilon)|p_j|$ since both r_j and p_j are in the same group $G^v_{i,g}$. We might not be able to fit all items $r_1, \ldots, r_m$ into a bin of size Q because $\sum_{i=1}^{m} |r_i| \leq (1+\epsilon) \sum_{i=1}^{m} |p_i| \leq (1+\epsilon)|\mathcal{T}| \leq (1+\epsilon)Q$. However, if we used two bins of size Q, we can pack the items into both bins without exceeding the capacity of either bin such that each item r_i is completely in one bin. Since $\mathcal{T}_1$ and $\mathcal{T}_2$ are not assigned to any lower level, they have not been used to cover any tokens so far in our algorithm and they both have unused capacity Q. Using the bin packing analogy, we could split $r_1, \ldots, r_m$ between $\mathcal{T}_1$ and $\mathcal{T}_2$. We could assign $r_1, \ldots, r_j$ (for the maximum j) to $\mathcal{T}_1$ such that $\sum_{i=1}^{j} |r_i| \leq Q$ and the rest, $r_{j+1}, \ldots, r_m$ to $\mathcal{T}_2$. Since $\sum_{i=1}^{m} |r_i| \leq (1+\epsilon)Q$, we can ensure we can distribute the tokens in r_i's amongst $\mathcal{T}_1$ and $\mathcal{T}_2$ such that both $\mathcal{T}_1$ and $\mathcal{T}_2$ cover at most Q tokens. Although there are two copies of each partial tour p_i in $X_{v,i}$, according to our approach, we are using at most one of them (their coverage would be zero if they are not used). If the coverage of one of the extra partial tours is non-zero, we also showed that if it picks up tokens from a partial tour in $G^v_{i,g}$, it would pick up exactly $h^{v,\max}_{i,g}$ tokens, proving the 2nd property of the Lemma.

Also, note that for each partial tour $r_k \in G^v_{i,g}$ and for each level $\ell' > \ell$ if r_k visits a vertex $v' \in V_{\ell'}$, then the partial tour of r_k at $T_{v'}$ already satisfies the

properties that: either its size belongs to a small vertex-bucket pair (v', b_i) (so has one of $O(\log^3 n/\epsilon)$ many possible values) or if it belongs to a big vertex bucket pair $(v', b_{i'})$ then its size is equal to $h_{i',j'}^{v',max}$ for some group j' and hence one of $O((\log Q \log n)/\epsilon^2)$ possible values. This implies that for the extra tours of $X_{v,i}$, after we reassign partial tours of $G_{i,g}^v$ to them (to cover the orphant nodes), each will have a size exactly equal to $h_{i,g}^{v,max}$ at level ℓ and at lower levels $V_{>\ell}$, the tours either belong to a small or a big bucket. Since a partial tour in a big vertex-bucket pair has one of $g = (2\delta \log n)/\epsilon^2$ many sizes and a partial tour in small vertex-bucket pair has one of $O(\log^3 n/\epsilon)$ tour sizes. Each tour in a vertex-bucket pair could have at most $\max\{O(\log n/\epsilon^2), O(\log^3 n/\epsilon)\} = O(\log^3 n/\epsilon^2)$ many tour sizes and since there are $O(\log Q/\epsilon)$ many buckets, the size of partial tours of $X_{v,i}$ at a vertex at level $\ell' > \ell$ is one of $O(\log Q \log^3 n/\epsilon^3)$ many possible sizes. This establishes the 3rd property of the lemma. ■

Therefore, using Lemma 3, all the tokens of T_v remain covered by partial tours; those partial tours in $G_{i,j}^v$ (for $1 \leq j < g$) are tied to the top parts of the tours from group $G_{i,j+1}^v$ and the partial tours of $G_{i,g}^v$ will be tied to extra tours designated to level ℓ. We also add extra tokens at v to be picked up by the partial tours of T_v so that each partial tour has a size exactly equal to the maximum size of a group. All in all, the extra cost paid to build OPT_ℓ (from $\mathrm{OPT}_{\ell+1}$) is for the extra tours designated to level ℓ.

Theorem 7 ***(Structure Theorem)*** *Let* OPT *be the cost of the optimal solution to instance $\mathcal{I}$. We can build an instance $\mathcal{I}'$ on the same tree T such that each node has ≥ 1 tokens and there exists a near-optimal solution* OPT' *for $\mathcal{I}'$ having cost* $(1+4\epsilon)$OPT *w.h.p with the following property. The partial tours going down subtree T_v for every node v in* OPT' *has one of $O((\log Q \log^3 n)/\epsilon^3)$ possible sizes. More specifically, suppose (v, b_i) is a bucket pair for* OPT'*. Then either:*

- *b_i is a small bucket and hence there are at most $\alpha \log^3 n/\epsilon^2$ many partial tours of T_v whose size is in bucket b_i, or*
- *b_i is a big bucket; in this case there are $g = (2\delta \log n)/\epsilon^2$ many group*

sizes in b_i: $\sigma_i \leq h_{i,1}^{v,max} \leq \ldots \leq h_{i,g}^{v,max} < \sigma_{i+1}$ and every tour of bucket i has one of these sizes.

Proof. We will show how to modify OPT to a near-optimal solution OPT′. We start from $\ell = h$ and let $\mathrm{OPT}_\ell = \mathrm{OPT}$. For decreasing values of ℓ we show, for each ℓ how to modify OPT_{l+1} to obtain OPT_ℓ. We do this in the following manner: we do not modify partial tours in small buckets. However, for tours in big buckets, in each vertex/bucket pair (v, b_i) in level $\ell - 1$, we place them into g groups $G_1^v, \ldots, G_g^v$ of equal sizes by placing the i'th n_v/g partial tours into G_i^v. We have a mapping f from each partial tour in G_{i-1}^v to one in G_i^v for $i \in \{2, \ldots, g\}$. We modify OPT_ℓ to $\mathrm{OPT}_{\ell+1}$ in the following way: for each tour $\mathcal{T}$ that has a partial tour $t \in G_i^v$, replace the bottom part of $\mathcal{T}$ at v from t to $f(t)$ (which is in G_{i-1}^v). For each tour $t \in G_{i-1}^v$, we will add $h_{i-1}^{v,\max} - |t|$ many extra tokens at v. Note that by this change, the size of any tour such as $\mathcal{T}$ can only decrease and we are not violating feasibility of the tour because $h_{i-1}^{v,\max} \leq h_i^{v,\min}$. However, the tokens in T_v picked up by the partial tours in $G_{i,g}^v$ are not covered by any tours. We can use Lemma 3 to show how we can use extra tours to cover the partial tours in $G_{i,g}^v$ such that the new partial tours have size exactly $h_{i,g}^{v,\max}$.

We will inductively repeat this for levels $\ell - 2, \ell - 3, \ldots, 1$ and obtain $\mathrm{OPT}_1 = \mathrm{OPT}'$. Note that by adding extra tokens $h_{i-1}^{v,\max} - |t|$ for a tour $t \in G_{i-1}^v$, we are enforcing that the coverage of each tour is the maximum size of tours in its group. In a big bucket, there are $g = (2\delta \log n)/\epsilon^2$ many group sizes, so there are $O(\log n/\epsilon^2)$ possible sizes for tours in big buckets at a node. In a small bucket, there can be at most $\alpha \log^3 n/\epsilon^2$ many tours and since there are $\tau = O(\log Q/\epsilon)$ many buckets, there can be at most $O((\log Q \log^3 n)/\epsilon^3)$ many tour sizes covering T_v.

Using Lemma 1, we know the cost of the extra tours is at most $4\epsilon \cdot \textsc{opt}$ with high probability, so the cost of $\textsc{opt}' \leq (1 + 4\epsilon)\textsc{opt}$. ∎

2.3 Dynamic Program

In this section we complete the proof of Theorem 2. We will describe how we can compute a solution of cost at most $(1+4\epsilon)\textsc{opt}$ using dynamic programming and based on the existence of a near-optimum solution guaranteed using the structure theorem. For each vertex/bucket pair, we do not know if the bucket is small or big, so we will consider subproblems corresponding to both possibilities. Informally, we will have a vector $\vec{n} \in [n]^\tau$ where if $i < 1/\epsilon$, n_i keeps track of the exact number of tours of size i and for $i \geq 1/\epsilon$, $\vec{n}_i$ keeps track of the number of tours in bucket b_i, or tours covering between $[\sigma_i, \sigma_{i+1})$ tokens. Let o_v denote the total number of tokens to be picked up across all nodes in the subtree T_v. Since each node has at least one token, $o_v \geq |V(T_v)|$. We will keep track of three other pieces of information conditioned on whether b_i is a small or big bucket. If b_i is a small bucket, we will store all the tour sizes exactly. Since the number of tours in a small bucket is at most $\gamma = \alpha \log^3 n/\epsilon^2$, we will use a vector $\vec{t}^{\,i} \in [n]^\gamma$ to represent the tours of a small bucket where $\vec{t}^{\,i}_j$ represents the size of j-th tour in bucket b_i. Suppose b_i is a big bucket, there are $g = (2\delta \log n)/\epsilon^2$ many tour sizes in the bucket corresponding to n^g possibilities. For each big bucket b_i at node v, we need to keep track of the following information,

- $\vec{h}^{\,i}_v \in [n]^g$ is a vector where $\vec{h}^{\,i}_{v,j} = h^{v,\max}_{i,j}$, which is the size of the maximum tour in group j of bucket i at node v.
- $\vec{l}^{\,i}_v \in [n]^g$ is a vector where $\vec{l}^{\,i}_{v,j}$ denotes the number of partial tours covering $h^{v,\max}_{i,j}$ tokens which lies in group j of bucket i at node v.

Let $\vec{y}_v$ denote a configuration of tours across all buckets of v.

$$\vec{y}_v = [o_v, \vec{n}_v, (\vec{t}^{\,1}_v, \vec{h}^{\,1}_v, \vec{l}^{\,1}_v), (\vec{t}^{\,2}_v, \vec{h}^{\,2}_v, \vec{l}^{\,2}_v), \ldots, (\vec{t}^{\,\tau}_v, \vec{h}^{\,\tau}_v, \vec{l}^{\,\tau}_v)].$$

Note that a bucket b_i is either small or big and cannot be both, hence given $(\vec{t}^{\,i}_v, \vec{h}^{\,i}_v, \vec{l}^{\,i}_v)$, it cannot be the case that $\vec{t}^{\,i}_v \neq \vec{0}, \vec{h}^{\,i}_v \neq \vec{0}$ and $\vec{l}^{\,i}_v \neq \vec{0}$. The subproblem $\mathbf{A}[v, \vec{y}]$ is supposed to be the minimum cost collection of partial tours going down T_v (to cover the tokens in T_v) and the cost of using the parent

edge of v having tour profile corresponding to $\vec{y}$. Our dynamic program heavily relies on the properties of the near-optimal solution in the structure theorem. Let v be a node. We will compute $A[\cdot,\cdot]$ in a bottom-up manner, computing $\mathbf{A}[v,\vec{y}_v]$ after we have computed the entries for the children of v.

The final answer is obtained by looking at the various entries of $\mathbf{A}[r,\cdot]$ and taking the smallest one. First, we argue why this will correspond to a solution of cost no more than OPT$'$. We will compute our solution in a bottom-up manner.

For the base case, we consider leaf nodes. A leaf node v with parent edge e could have $o_v \geq 1$ tokens at v. We will set $\mathbf{A}[v,\vec{y}_v] = 2{\cdot}w(e){\cdot}m_v$ where m_v is the number of tours in $\vec{y}_v$ if the total sum of tokens picked up by the partial tours in $\vec{y}_v$ is exactly o_v. Recall that $f(e)$ is the load on (i.e. number of tours using) edge e. From our structure theorem, we know there exists a near optimum solution such that each partial tour of T_v has one of $O((\log Q \log^3 n)/\epsilon^3)$ tour sizes and for each small bucket, there are at most $\alpha \log^3 n/\epsilon^2$ partial tours in it. For every big bucket, there are $g = (2\delta \log n)/\epsilon^2$ many group sizes and every tour of bucket i has one of these sizes. The base case follows directly from the structure theorem.

To compute cell $\mathbf{A}[v,\vec{y}_v]$, we would need to use another auxiliary table $\mathbf{B}$. Suppose v has k children $u_1,\ldots,u_k$ and assume we have already calculated $\mathbf{A}[u_j,\vec{y}]$ for every $1 \leq j \leq k$ and for all vectors $\vec{y}$. Then we define a cell in our auxiliary table $\mathbf{B}[v,\vec{y}'_v,j]$ for each $1 \leq j \leq k$ where $\mathbf{B}[v,\vec{y}'_v,j]$ is the minimum cost of covering $T_{u_1} \cup \ldots \cup T_{u_j}$ where $\vec{y}'_v$ is the tour profile for the union of subtrees $T_{u_1} \cup \ldots \cup T_{u_j}$. In other words, $\mathbf{B}[v,\vec{y}'_v,j]$ is what $\mathbf{A}[v,\vec{y}_v]$ is supposed to capture when restricted only to the first j children of v. We will set $\mathbf{A}[v,\vec{y}_v] = \mathbf{B}[v,\vec{y}'_v,k] + 2{\cdot}w(e){\cdot}m_v$ where m_v is the number of different tours in $\vec{y}'_v$. We will assume the parent edge of the depot has weight 0. Suppose T_{u_i} has o_i tokens, then the number of tokens in T_v is at least $1+\sum_{i=1}^{k} o_i$. To compute entries of $\mathbf{B}[v,\cdot,\cdot]$, we use both $\mathbf{A}$ and $\mathbf{B}$ entries for smaller subproblems of v in the following way:

Case 1: j = 1: This is the case when we restrict the coverage to only the

first child of v, u_1.

$$\mathbf{B}[v, \vec{y}'_v, 1] = \min_{\vec{y}'} \left\{\mathbf{A}[u_1, \vec{y}']\right\}$$

We will find the minimum cost configurations $\vec{y}'$ such that $\vec{y}'_v$ and $\vec{y}'$ are consistent with each other. We say $\vec{y}'_v$ and $\vec{y}'$ are consistent if a tour in $\vec{y}'_v$ either only covers tokens at v and does not visit any node below v or $\vec{y}'_v$ consists of a tour from $\vec{y}'$ plus zero or more extra tokens picked up at v. Moreover, every tour in $\vec{y}'$ is part of some tour in $\vec{y}'_v$.

Case 2: $2 \leq j \leq k$. We will assume we have computed $\mathbf{B}[v, \vec{y}', j-1]$ and $\mathbf{A}[u_j, \vec{y}'']$ and we have

$$\mathbf{B}[v, \vec{y}'_v, j] = \min_{\vec{y}', \vec{y}''} \{\mathbf{B}[v, \vec{y}', j-1] + \mathbf{A}[u_j, \vec{y}'']\}.$$

There are four possibilities for each partial tour t_v at node v going down T_v covering tokens for subtrees rooted at children $u_1, \ldots, u_k$.

- t_v could be a tour that only picks up tokens at v and does not pick up tokens from subtrees $T_{u_1} \cup \ldots \cup T_{u_j}$.
- t_v could be a tour that picks up tokens at v and picks up tokens only from subtrees $T_{u_1} \cup \ldots \cup T_{u_{j-1}}$.
- t_v could be a tour that picks up tokens at v and picks up tokens only from subtree T_{u_j}.
- t_v could be a tour that picks up tokens at v and picks up tokens from subtrees $T_{u_1} \cup \ldots \cup T_{u_j}$.

We would find the minimum cost over all configurations $\vec{y}'_v, \vec{y}'$ and $\vec{y}''$ as long as $\vec{y}'_v, \vec{y}'$ and $\vec{y}''$ are consistent. We say tours $\vec{y}'_v, \vec{y}'$ and $\vec{y}''$ are consistent if there is a way to combine partial tours from $\vec{y}'$ and $\vec{y}''$ to form a partial tour in $\vec{y}'_v$ while also picking up extra tokens at node v. We will define consistency more rigorously in the next section.

2.3.1 Checking Consistency

In our dynamic program, for the inner DP, we are given three vector $\vec{y}'_v, \vec{y}', \vec{y}''$ where v is a node having children $u_1, \ldots, u_j$. $\vec{y}'$ represents the configuration of tours in $T_{u_1} \cup \ldots \cup T_{j-1}$ and $\vec{y}''$ represents the configuration of tours covering T_{u_j}. For the case of checking consistency for case 1, we will assume $\vec{y}'' = \vec{0}$. Suppose we are given o_v (for node v), o_u for children $u_1, \ldots, u_{j-1}$, and o_w for u_j, we can infer that there are $o'_v = o_v - o_u - o_w$ extra tokens that need to be picked at v. o'_v tokens need to be distributed amongst tours in $\vec{y}_v$. There are four possibilities for each tour t_v in $\vec{y}'_v$.

- t_v could be a tour that picks up extra tokens at v and picks up tokens only from subtrees $T_{u_1} \cup \ldots \cup T_{u_{j-1}}$.
- t_v could be a tour that picks up extra tokens at v and picks up tokens only from subtree T_{u_j}.
- t_v could be a tour that picks up extra tokens at v and picks up tokens from subtrees $T_{u_1} \cup \ldots \cup T_{u_j}$.

For simplicity, we will refer to a tour picking up tokens in $T_{u_1} \cup \ldots \cup T_{u_{j-1}}$ to be t_u and a tour picking up tokens from T_{u_j} to be t_w.

Definition 6 *We say configurations $\vec{y}'_v, \vec{y}'$ and $\vec{y}''$ are **consistent** if the following holds:*

- *Every tour in $\vec{y}'$ maps to some tour in $\vec{y}'_v$.*
- *Every tour in $\vec{y}''$ maps to some tour in $\vec{y}'_v$.*
- *Every tour in $\vec{y}'_v$ has at most two tours mapping to it and both tours cannot be from $\vec{y}'$ or $\vec{y}''$.*
- *Suppose only one tour (t_u) maps to a tour t_v in $\vec{y}'_v$. The number of extra tokens picked up by tour t_v at v is $|t_v| - |t_u|$.*
- *Suppose t_v, a tour in $\vec{y}'_v$ has two tours: t_u in $\vec{y}'$ and t_w in $\vec{y}''$ mapped to it, then the number of extra tokens picked up by tour t_v at v is $|t_v| - |t_u| - |t_w|$.*

- *The extra tokens at v, $o'_v = o_v - o_u - o_w$, are picked up by the tours in $\vec{y}'_v$.*

Consistency ensures that we can patch up tours from subproblems and combine them into new tours in a correct manner while also picking up extra tokens at v. Now we will describe how we can compute consistency. Let $\vec{z}$ be a vector containing a subset of information contained in $\vec{y}$.

$$\vec{z}_v = [\vec{n}_v, (\vec{t}_v^1, \vec{h}_v^1, \vec{l}_v^1), (\vec{t}_v^2, \vec{h}_v^2, \vec{l}_v^2), \ldots, (\vec{t}_v^\tau, \vec{h}_v^\tau, \vec{l}_v^\tau)].$$

From now on, we will choose to not write $\vec{n}_v$ explicitly since we can figure out the entries of the vector from $\vec{l}$. Suppose $|t_v|$ is the length of a tour in $\vec{z}'_v$. Let $\vec{z}'_v - t_v$ refer to the configuration $\vec{z}'_v$ having one less tour of size $|t_v|$. Let $\mathbf{C}[o'_v, \vec{z}'_v, \vec{z}', \vec{z}''] =$ True if it is consistent and False otherwise. For the base case, $\mathbf{C}[0, \vec{0}, \vec{0}, \vec{0}] =$True. For the recurrence, we will look at all possible ways of combining $\vec{z}'$ and $\vec{z}''$ into $\vec{z}'_v$ while also picking up extra tokens o'_v. Note that t_v is always non-zero, but both or one of t_u or t_w could be zero.

$$\mathbf{C}[o'_v, \vec{z}'_v, \vec{z}', \vec{z}''] = \bigvee_{\substack{t_v, t_u, t_w \\ |t_v| = |t_u| + |t_w| + o_c}} \mathbf{C}[o'_v - o_c, \vec{z}'_v - t_v, \vec{z}' - t_u, \vec{z}'' - t_w].$$

2.3.2 Time Complexity

We will work bottom-up and assume we have already pre-computed our consistency table. Computing $\mathbf{B}[\cdot, \cdot, \cdot]$ requires looking at previously computed $\mathbf{B}[\cdot, \cdot, \cdot]$ and $\mathbf{A}[\cdot, \cdot]$. Given $\vec{y}'_v, \vec{y}'$ and $\vec{y}''$ which are all consistent, computing the cost of $\vec{y}'_v$ using $\vec{y}'$ and $\vec{y}''$ takes $O(1)$ time. Each $\vec{y}'_v$ consists of

1. $\vec{n}$ has $n^{O(\log n/\epsilon)}$ possibilities.
2. Each $\vec{t}^i$ has $n^{O(\log^3 n/\epsilon^2)}$ possibilities since there are $O(\log^3 n/\epsilon)$ tours in a small bucket.
3. Each $\vec{h}$ and $\vec{l}$ have $n^{O(g)}$ possibilities. Recall that $g = (2\delta \log n)/\epsilon^2$, so each $\vec{h}$ and $\vec{l}$ have $n^{O(\log n/\epsilon^2)}$ possibilities.
4. Each triple $(\vec{t}^i, \vec{h}^i, \vec{l}^i)$ has $n^{O(\log^3 n/\epsilon^2)}$ possibilities.

5. $(\vec{t}^1, \vec{h}^1, \vec{l}^1), (\vec{t}^2, \vec{h}^2, \vec{l}^2), \ldots, (\vec{t}^\tau, \vec{h}^\tau, \vec{l}^\tau)$ have $n^{O(\tau \log^3 n/\epsilon^2)} = n^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities since $\tau = O(\log Q/\epsilon)$.

In total, each $\vec{y}'_v$ has $n^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities. For each $\vec{y}'_v$, we will have $n^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities for $\vec{y}_u$ and $\vec{y}_w$. Since there are $n^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities for $\vec{y}'_v$, the cost of computing the DP entries for a single node v would be $n^{O((\log Q \log^3 n)/\epsilon^3)}$ and since there are n nodes in the tree, the total time of computing the DP table assuming the consistency table is precomputed is $n^{O((\log Q \log^3 n)/\epsilon^3)}$.

Before we compute our DP, we will first compute the consistency table $\mathbf{C}[\cdot, \cdot, \cdot, \cdot]$. Similar to our DP table, each entry of the consistency table has $n^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities. Assuming we have already precomputed smaller entries of $\mathbf{C}$, there are $n^{O((\log Q \log^3 n)/\epsilon^3)}$ ways of picking t_v, t_u and t_w. For a fixed $\vec{y}_v, \vec{y}_u, \vec{y}_w$ and o'_v, computing $\mathbf{C}[o'_v, \vec{z}'_v, \vec{z}', \vec{z}'']$ takes $n^{O((\log Q \log^3 n)/\epsilon^3)}$ time. Since there are only $n^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities for $\vec{z}'_v, \vec{z}'$ and $\vec{z}''$, the cost of computing all entries of the consistency table is $n^{O((\log Q \log^3 n)/\epsilon^3)}$.

The time for computing both the DP table and consistency table is $n^{O((\log Q \log^3 n)/\epsilon^3)}$, so the total time taken by our algorithm is $n^{O((\log Q \log^3 n)/\epsilon^3)}$. For the unit demand case, since $Q \leq n$, the runtime of our algorithm is $n^{O(\log^4 n/\epsilon^3)}$.

2.4 Extension to Splittable CVRP

We can extend our algorithm for unit demand CVRP in trees and show how we can get a QPTAS for splittable CVRP as long as the demands are quasi-polynomially bounded (Corollary 1). In our algorithm for unit demand CVRP, we viewed the demand of each node as a token placed at the node. For splittable CVRP, we could assume each node has $1 \leq d(v) < nQ$ tokens and we can use the same structure theorem as before by modifying tours such that there are at most $O((\log Q \log^3 n)/\epsilon^3)$ different tour sizes for partial tours at a node. We can use the same DP to compute the solution. Each $\vec{y}_v$ consists of:

1. $\vec{n}$ which has $(nQ)^{O(\log n/\epsilon)}$ possibilities.

2. Each $\vec{t}^i$ has $(nQ)^{O(\log^3 n/\epsilon^2)}$ possibilities since there are $O(\log^3 n/\epsilon)$ tours in a small bucket.

3. Each $\vec{h}$ and $\vec{l}$ have $(nQ)^{O(g)}$ possibilities. Recall that $g = (2\delta \log n)/\epsilon^2$, so each $\vec{h}$ and $\vec{l}$ have $(nQ)^{O(\log n/\epsilon^2)}$ possibilities.

4. Each triple $(\vec{t}^i, \vec{h}^i, \vec{l}^i)$ has $(nQ)^{O(\log^3 n/\epsilon^2)}$ possibilities.

5. $(\vec{t}^1, \vec{h}^1, \vec{l}^1), (\vec{t}^2, \vec{h}^2, \vec{l}^2), \ldots, (\vec{t}^\tau, \vec{h}^\tau, \vec{l}^\tau)$ have $(nQ)^{O(\tau \log^3 n/\epsilon^2)} = (nQ)^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities since $\tau = O(\log Q/\epsilon)$.

Similar to the analysis of the runtime of the unit demand case, the time complexity of computing the entries of DP tables $\mathbf{A}, \mathbf{B}$, and the consistency table $\mathbf{C}$ is, $(nQ)^{O((\log Q \log^3 n)/\epsilon^3)}$. Suppose $Q = n^{O(\log^c n)}$, then the runtime of our algorithm is $n^{O(\log^{2c+4} n/\epsilon^3)}$.

2.5 Height reduction

In this section, we will prove Theorem 6. The first goal is to decompose the edge set of the tree T into edge-disjoint paths. We will do so using the following lemma, similar to Lemma 5 from Cygan et al. [15] to obtain such a decomposition in polynomial-time for a different problem.

Lemma 4 *There exists a decomposition of the edge set of T into edge-disjoint paths which can be grouped into $s = O(\log n)$ collections (called levels) $L_1, \ldots, L_s$ such that the following hold:*

1. *A root-to-leaf path P in T can be written as $P = Q_0 Q_1 \ldots Q_s$ where Q_i is either a path in L_i or it is empty.*

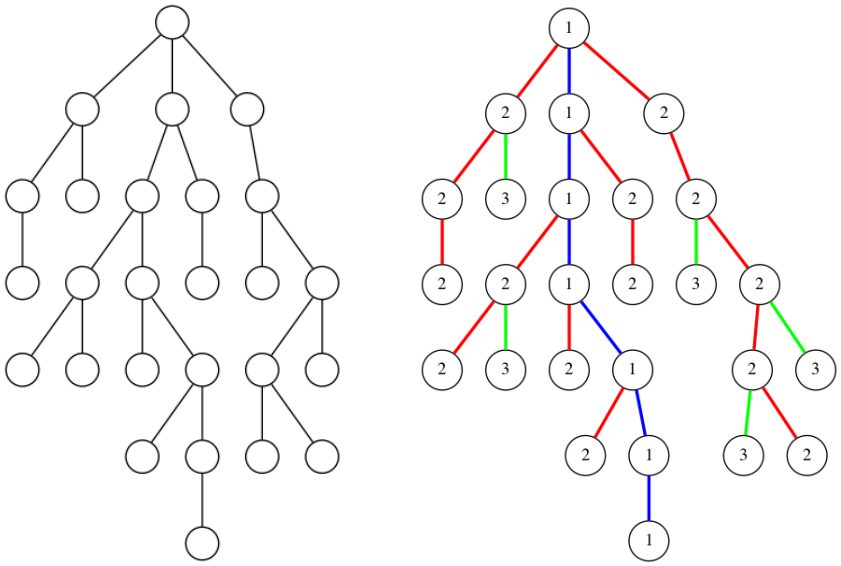
2. *P would use a path from a lower level L_i before using a path from a higher level, L_j where $i < j$.*

Proof. Given a tree T, a D-path of T is a root-to-leaf path $P = v_1 v_2 \ldots v_k$ such that v_{i+1} is the child of v_i with the largest number of nodes in the tree rooted at $T_{v_{i+1}}$. If there are multiple children with the same number of descendants,

break ties arbitrarily. Let P be a D-path. All the nodes in D-path P receive label 1. Let $T_1, \ldots, T_c$ be the set of trees obtained from $T - P$. Let P_i be the D-path for T_i. We will label all nodes in P_i to be 2. We will repeat this process recursively by finding D-paths for trees resulting from $T_i - P_i$ and labelling every node in the D-path with value corresponding to the depth of recursion. Each step involves finding a D-path, labelling the nodes of the path, deleting the path and recursively repeating the process for the resulting trees (with the value of the label increased by 1). Nodes of D-paths of trees at depth ℓ in the recursion receive labels ℓ. We will terminate this process when all nodes have been labelled. Let L_j denote the collection of all D-paths whose nodes received the label j (see Figure 2.1).

Note that after the first step, the trees $T_1, \ldots, T_c$ satisfy the property that $|V(T_i)| \leq \frac{|V(T)|}{2}$ i.e., each tree is at most half of the original tree. This is because we pick the child with the largest number of nodes in the subtree rooted at it. After each step, the size of the new components formed is at most half the size of the previous component, hence we would use at most $\log n$ labels to label all nodes in the tree. ■

The following is an example of such labelling where each color represents a level.



(a) A tree before labelling.

(b) Blue edges are level 1, red edges are level 2 and green edges are level 3.

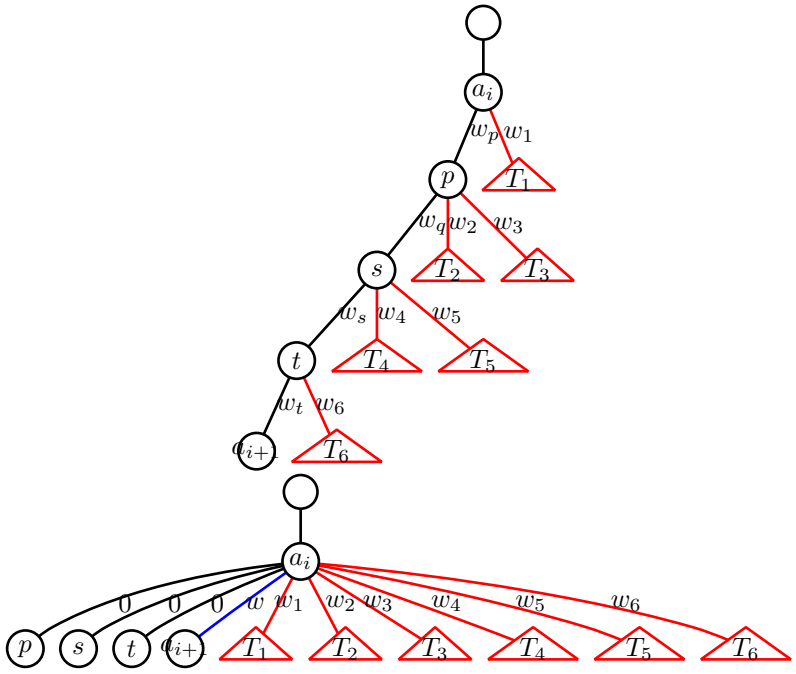
Figure 2.1: An example of a tree before and after applying labels to nodes

2.5.1 Creating a new tree

Given a tree T, we can use Lemma 4 to decompose the tree into edge-disjoint paths. Next, we describe an algorithm to modify the tree recursively into a low height tree. The first step is to look at all the paths in L_1. L_1 is a special case since there is only one path in L_1 which goes from the depot to a leaf node. All the other levels L_i could have multiple disjoint paths. Let P be the path in L_1 and let $l(P)$ be the number of edges in path P. If $l(P) \leq \frac{\delta \log n}{\epsilon}$ for a $\delta > 0$ to be specified, then we are done for L_1.

However, if $l(P) > \frac{\delta \log n}{\epsilon}$, we will compress the path into a low height one. We will do a sequence of what is called up-pushes. We will pick $s \leq \frac{\delta \log l(P)}{\epsilon}$ points to be **anchor points**. Let us call the anchor points $a_1, \ldots, a_s$ where a_1 is the anchor point closest to the root and a_s is closest to the leaf. We will later show how to find these anchor points.

Figure 2.2: A tree before an up-push (top) and after (bottom) with reduced height. The blue edge connecting a_i and a_{i+1} has weight $w = w_p + w_q + w_s + w_t$



Each up-push acts on nodes in P between two consecutive anchor points a_i, a_{i+1} of the path P. During an up-push, we take all nodes in P that lie between a_i and a_{i+1}, which we will call P', and make each node in P' a child of a_i with the edge connecting them to a_i having weight 0. Suppose there is a child subtree T_j, which is a child of a node in P' with edge connection cost w_j, the subtree T_j will become a child of a_i with the edge connecting them having cost w_j (see Figure 2.2). Once we have completed up-pushes for all paths in L_1, we will find anchor points and perform up-pushes for each path in L_2. We will repeat this for paths in L_i after our algorithm has finished up-pushes for paths in L_{i-1}.

We will now describe how we can find the anchor points. We will first describe what we would like to achieve from anchor points. We want the cost associated with a path in L_i for some tour to differ by at most $O(\epsilon)$ in our new tree compared to the the original tree. Suppose P is a path in L_i and a tour t is travelling P down to node u which is between a_i and a_{i+1}. Then the cost of the portion of the tour from root of P to a_i is the same in the original tree and the new tree; however the cost to travel from a_i to u is zero. We would

like this cost in the original tree to be a small factor of the cost from the root of P to a_i.

Our algorithm works as follows from top to bottom. For any path P in L_i, we will set the top node of the path to be a_1 and its child in P to be a_2. Our goal is to pick a_i and a_{i+1} for $i > 2$ such that $w(a_i, a_{i+1}) > \epsilon \cdot w(a_1, a_i)$ and $w(a_i, v) \leq \epsilon \cdot w(a_1, a_i)$ where v is the last vertex on a_i, a_{i+1} path before a_{i+1}. If there is no a_{i+1} such that $w(a_i, a_{i+1}) > \epsilon \cdot w(a_1, a_i)$, then we set the last node of P to be a_{i+1}. So, we pick a_{i+1} to be the farthest vertex from a_i in P such that $w(a_i, v) \leq \epsilon \cdot w(a_1, a_i)$ where v is the last node before a_{i+1}. This in turn would imply that $w(a_1, a_{i+1}) > (1 + \epsilon)w(a_1, a_i)$, except if a_{i+1} is the last node of the path. Hence, $w(a_1, a_i) > (1 + \epsilon)^{i-2} w(a_1, a_2) > (1 + \epsilon)^{i-2}$. Since edge weights are at most $2n^3/\epsilon^2$, the number of anchor points are at most $\frac{\delta \log n}{\epsilon}$ for some constant $\delta > 0$

2.5.2 Analysis

In the last section, we showed that every path in some level L_i can be made to have at most $O\left(\frac{\log n}{\epsilon}\right)$ nodes.

Lemma 5 *The height of the new tree is* $O\left(\frac{\log^2 n}{\epsilon}\right)$.

Proof. In our algorithm, we first decomposed the tree into a set of edge-disjoint paths. The decomposition guarantees that one would first visit a lower level node in any root-to-leaf path before visiting one with a higher level. Since there are at most $O(\log n)$ different levels, any root-to-leaf path will be a disjoint union of paths from levels $L_1, \ldots, L_s$ and there can be at most one path from each level. Since the height of a path in any level, L_i is at most $O\left(\frac{\log n}{\epsilon}\right)$, and there are at most $O(\log n)$ different levels, the maximum height in our new tree is at most $O\left(\frac{\log^2 n}{\epsilon}\right)$ ∎

Suppose we take a path P at some level L_c. Let us fix a tour in an optimal solution and let the farthest point in P the tour travels to be between anchor points $[a_i,a_{i+1})$. We use $[a_i,a_{i+1})$ denote that the tour crosses a_i but will not cross a_{i+1}. Let T be the original tree and let T' be the new tree with reduced height. A tour in the optimal solution for T' can visit nodes lying between a_i

and a_{i+i} at no additional cost after visiting a_i. Suppose the cost of traversing the edges of P in T' is denoted by d, then the cost of traversing the edges of P in T is going to be at most $(1 + O(\epsilon))d$. This is because the cost of the edges between a_i and the vertex before a_{i+1} sum to at most $O(\epsilon)w(r, a_i)$. Hence, the additional cost to cover them in T is only going to be at most an ϵ fraction more.

Lemma 6 *Let T be the original tree, T' be the new tree,* OPT *be the cost of the optimal set of tours covering T, and* OPT$'$ *be the cost of the optimal set of tours covering T'. Then,*

$$\text{OPT}' \leq \text{OPT} \leq (1 + \epsilon)\text{OPT}'.$$

Proof. Let us fix an optimal set of tours covering tree T with cost OPT. Suppose we pick a tour t and decompose this tour into paths each of which is entirely within one level L_i. Suppose P is a path of t in some level L_c. Let the farthest point in P the tour travels to be between anchor points $[a_i,a_{i+1})$. In our construction, the cost to visit any point lying between the root of P and a_i is the same in both T and T'. However, in T', the tour can visit any node lying between a_i and a_{i+1} for free, but the tour would have an additional cost to traverse these edges in tree T. Hence, for any path such P, the cost of a tour t to traverse edges in P is less in T' compared to T. Since any tour costs no more in instance T', we have OPT$' \leq$ OPT.

Conversely, the extra cost of covering points lying between a_i and a_{i+1} in T is at most $O(\epsilon)$ times the cost of path P (based on the property of anchor points). So the cost of using a path like P is at most an ϵ factor more in T compared to T'. Thus, the cost of any tour t in T is at most $1 + \epsilon$ times the cost of the same tour in T' and hence OPT $\leq (1 + \epsilon)$OPT$'$ ∎

Instead of T, we can solve the instance on T' with height $O(\log^2 n/\epsilon)$ and lift the solution for T' back to a solution for T. We obtain a solution for T with cost at most $(1 + \epsilon)$OPT.

Chapter 3

QPTAS for CVRP in Bounded Treewidth Graphs

In this chapter, we will use ideas from QPTAS for CVRP in trees and extend it to obtain a QPTAS for CVRP in Bounded Treewidth graphs.

3.1 Problem Overview

Given a graph $G = (V, E)$ with treewidth k, we will assume we are given a tree decomposition $T = (V', E')$. We will refer to G as the graph and T as the tree. We will refer to vertices in V by **nodes** and vertices in V' by **bags**. We will refer to edges in E by **edges** and edges in E' by **superedges**. Recall the definition of a tree decomposition,

A **tree decomposition** of a graph G is a pair $(T, \{B_t\}_{t \in V(T)})$, where T is a tree whose every node $t \in V'$ is assigned a vertex subset $B_t \subseteq V(G)$, called a bag, such that the following three conditions hold:

1. $\cup_{t \in V(T)} B_t = V(G)$. In other words, every vertex of G is in at least one bag.

2. For every $uv \in E(G)$, there exists a node t of T such that bag B_t contains both u and v.

3. For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in B_t\}$, i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree

of T.

For a bag s, let C_s denote the union of nodes in bags below s including s. Bag s forms a boundary or border between nodes in C_s and $V(G)\backslash C_s$. We will assume an arbitrary bag containing the depot to be root of the tree decomposition. Let k be the treewidth of our graph G. We will assume that following properties hold for our tree decomposition T of G from the work of Boedlander and Hagerup [12],

- T is binary.
- T has depth $O(\log n)$.
- The width of T is at most $k' = 3k + 2$.

To simplify notation, by replacing k' with k we will assume T has height $\delta \log n$ for some fixed $\delta > 0$ and each bag has width k. From the third property of a tree decomposition, we know that for every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree of T. Since the bags associated with a node $u \in V(G)$ correspond to a subtree in T, we will place the demand/tokens of u at the root bag of the tree T_u i.e. the bag containing u closest to the root bag of T. Since T_u is a tree, we are guaranteed a unique root bag of T_u exists. We are doing this to ensure that the demand of a client is delivered exactly once.

3.1.1 Our Results

We will restate Theorem 3 and Corollary 2 for convenience.

Theorem 3 *For any $\epsilon > 0$, there is an algorithm that, for any instance of Unit Demand CVRP on bounded treewidth graphs (G, Q) where G is a graph with n vertices and treewidth k, outputs a solution of expected cost $(1+\epsilon)$ times the optimal solution in time $n^{O(k^2 \log^3 n/\epsilon^2)}$*

We will then show how we can extend our dynamic program to also compute a near-optimal solution for Splittable CVRP in graphs of bounded treewdith when $Q = n^{O(\log^c n)}$ to get the following corollary.

Corollary 2 *Let $\epsilon > 0$, there is an algorithm that, for any instance of the splittable capacitated vehicle routing problem on bounded treewidth graphs (G, Q) where G is a graph with n vertices, treewidth k and when $Q = n^{O(\log^c n)}$, outputs a solution of expected cost $(1+\epsilon)$ times the optimal solution in time $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$*

3.2 Structure Theorem

Similar to how we showed the existence of a near-optimum solution for trees, we will modify the optimum solution OPT in a bottom-up manner by modifying the tours covering the set of nodes below bag s, C_s. For each bag s, we change the structure of the partial tours going down C_s (by adding a few extra tours from the depot) and also adding some extra tokens for nodes in bag s so that the partial tours that visit C_s all have a size from one of polyogarithmic many possible sizes (buckets) while increasing the number and the cost of the tours by a small factor. Note that although a node can be in different bags, its initial demand is in one bag and we might add extra tokens to copies of it in other bags.

Similar to the case of tree, we assume the bags of the tree decomposition are partitioned into levels $V_1, \ldots, V_h$ where V_1 is the bag containing the depot and h is the height of T. For every tour $\mathcal{T}$ and every level ℓ, we can define the notion of top and bottom part similar to the case of trees. For every C_s, a tour $\mathcal{T}$ enters C_s through bag s using a node x and exists through node z where both x and z have to be in s. Note that x and z could be equal if the tour enters and exists s using the same node. For a bag s, let $n_s^{x,z}$ be the number of partial tours covering nodes in C_s that enter through x and exit through z in s. For each bag and entry/exit pair, we will define the notion of a small/big bucket similar to the case of trees. For a big bucket, we will place the $n_s^{x,z}$ tours (ordered by increasing size) into groups $G_1^{x,z,s}, \ldots, G_g^{x,z,s}$ of equal sizes. Let $h_i^{s,x,z,\max}(h_i^{s,x,z,\min})$ refer to the maximum (minimum) size of the tours in $G_i^{x,z,s}$.

Similar to the case of trees, let f be a mapping from a tour in $G_i^{x,z,s}$ to

one in $G_{i-1}^{x,z,s}$. Now suppose we modify OPT to OPT$'$ in the following way: for each tour $\mathcal{T}$ that has a partial tour in $t \in G_i^{x,z,s}$, replace the bottom part of $\mathcal{T}$ entering through x and exiting through z in s from t to $f(t)$ (which is in $G_{i-1}^{x,z,s}$). The only problem is that those tokens in C_s that were picked up by the partial tours in $G_g^{x,z,s}$ are not covered by any tours and like the case of trees, these are *orphant* tokens. For each tour $\mathcal{T}$ and its (new) partial tour $t \in G_i^{x,z,s}$, if we add $h_i^{x,z,s,\max} - |t|$ extra tokens at s to be picked up by t, then each partial tour has size exactly same as the maximum size of its group without violating the capacities. Similar to the case of trees, we will show that if $n_s^{x,z}$ is sufficiently large (at least polylogarithmic), then if we sample a small fraction of the tours of the optimum at random and add two copies of them (as extra tours), they can be used to cover the orphant tokens.

3.2.1 Changing OPT to a near-optimum structured solution

Similar to the structure theorem for trees, we will modify the optimal solution OPT to a near-optimum solution OPT$'$ having certain properties. We will start at the last level, and modify partial tours from OPT at level ℓ to obtain OPT_ℓ. We will then iteratively obtain $\mathrm{OPT}_{\ell-1}$ by modifying partial tours from OPT_ℓ at level $\ell-1$, and iteratively do this for each level until we obtain $\mathrm{OPT}_1 = \mathrm{OPT}'$.

Definition 7 *For a bag s, the i-th bucket, b_i, entering at x and exiting at z contains the number of tours of OPT_ℓ having coverage between $[\sigma_i, \sigma_{i+1})$ tokens in C_s where σ_i is the i-th threshold value. We will denote this by a entry/exit-bag-bucket configuration (s, b_i, x, z). Let $n_{s,i}^{x,z}$ be the number of tours in bucket b_i entering through x and exiting through z in bag s.*

Definition 8 *An entry/exit-bag-bucket configuration (s, b_i, x, z) is **small** if $n_{s,i}^{x,z}$ is at most $\alpha \log^2 n/\epsilon$ and is **big** otherwise, for a constant $\alpha \geq \max\{1, 20\delta\}$.*

Note that for any bag s and entry/exit-bag-bucket configuration (s, b_i, x, z), if (s, b_i, x, z) is small, we do not modify the partial tours in it. However, if

(s, b_i, x, z) is a big bucket, we create groups $G_{i,1}^{s,x,z}, \ldots, G_{i,g}^{s,x,z}$ of equal sizes, for $g = (2\delta \log n)/\epsilon$; so $|G_{i,j}^{s,x,z}| = \lceil n_{s,i}^{x,z}/g \rceil$. We also consider a mapping f (as before) which maps (in the same order) the tours $t \in G_{i,j}^{s,x,z}$ to the tours in $G_{i,j-1}^{s,x,z}$ for all $1 < j \leq g$. Consider set $\mathbf{T}_\ell$ of all the tours $\mathcal{T}$ in OPT_ℓ that visit a bag in one of the lower levels $V_{\geq \ell}$. Consider an arbitrary such tour $\mathcal{T}$ that has a partial tour t in a big entry/exit-bag-bucket configuration (s, b_i, x, z), suppose t belongs to group $G_{i,j}^{s,x,z}$. We replace t with $f(t)$ in $\mathcal{T}$.

Now, add some extra tokens at x to be picked up by $\mathcal{T}$ so that the size of the partial tour of $\mathcal{T}$ at C_s is exactly $h_{i,j-1}^{s,x,z,\max}$. If we make this change for all tours $\mathcal{T} \in \mathbf{T}_\ell$, each partial tour of them at level ℓ that was in a group $j < g$ of a big entry/exit-bag-bucket configuration (s, b_i, x, z) is replaced with a smaller partial tour from group $j-1$ of the same big entry/exit-bag-bucket configuration; after adding extra tokens to x at bag s (if needed), the size is the maximum size from group $j-1$. The tokens that were picked by partial tours from $G_{i,g}^{s,x,z}$ for a big entry/exit-bag-bucket configuration (s, b_i, x, z) are now orphant. We are going to (randomly) select a subset of tours of OPT as "extra tours" and add them to OPT' and modify them such that they cover all the tokens that are now orphant (i.e. those that were covered by partial tours of $G_{i,g}^{s,x,z}$ for all big entry/exit-bag-bucket configuration (s, b_i, x, z) at level ℓ). Suppose we select each tour $\mathcal{T}$ of OPT with probability ϵ. We make two copies of the *extra tour* and we designate both extra copies to bags at one of the levels V_ℓ that it visits with equal probability.

Lemma 7 *The expected cost of extra tours selected is* $2\epsilon \cdot \mathrm{OPT}$.

Proof. Suppose $f^+(e)$ and $f^-(e)$ denote the number of tours traveling edge e in each of the two directions. So the contribution of edge e to the optimal solution is $2 \cdot w(e) \cdot (f^+(e) + f^-(e))$; $\mathrm{OPT} = \sum_{e \in E} w(e) \cdot (f^+(e) + f^-(e))$. Let $m^+(e)$ ($m^-(e)$) denote the number of sampled tours from the tours contributing to $f^+(e)$ ($f^-(e)$). Since we used two extra copies for each sampled tour, the number of extra tours for an edge e is $2(m^+(e) + m^-(e))$. Let $\mathcal{T}_{e,1}, \ldots, \mathcal{T}_{e,f^+(e)+f^-(e)}$ be the tours using e in either directions. Like in the case of trees, it is possible for a tour to use edge e in both directions. Let $Y_{e,i}$ be a random variable which

is 1 if tour $\mathcal{T}_{e,i}$ is sampled and 0 otherwise.

$$\mathbb{E}[Y_{e,i}] = \mathbb{P}[\mathcal{T}_{e,i} \text{ is sampled}] = \epsilon.$$

Let $m^+(e) + m^-(e) = Y_e = \sum_{i=1}^{f^+(e)+f^-(e)} Y_{e,i}$. By linearity of expectations, we have

$$\mathbb{E}\big[m^+(e) + m^-(e)\big] = \mathbb{E}[Y_e] = \sum_{i=1}^{f^+(e)+f^-(e)} \mathbb{E}[Y_{e,i}] = \sum_{i=1}^{f^+(e)+f^-(e)} \epsilon = \epsilon\cdot(f^+(e)+f^-(e)).$$

Summing up the extra cost over all edges, the expected cost of the extra tours is

$$2\sum_{e\in E} \mathbb{E}\big[m^{in}(e) + m^{out}(e)\big] = 2\epsilon \cdot \sum_{e\in E}(f^+(e) + f^-(e)) = 2\epsilon \cdot \textsc{opt}.$$

■

Therefore, we can assume that the expected cost of all extra tours added is at most $2\epsilon \cdot \textsc{opt}$. Let X_ℓ be the set of extra tours designated to bags in level ℓ. We assume we add X_ℓ when we are building OPT_ℓ (it is only for the sake of analysis). For each bag $s \in V_\ell$ and entry/exit-bag-bucket configuration (s, b_i, x, z), let $X_i^{s,x,z}$ be those in X_ℓ whose partial tour in C_s has a size in bucket b_i. Each extra tour in X_ℓ will not be picking any of the tokens in levels $V_{<\ell}$ (as they will be covered by the tours already in OPT_ℓ); they are used to cover the orphant tokens created by partial tours of $G_{i,g}^{s,x,z}$ for each big entry/exit-bag-bucket configuration (s, b_i, x, z) with $s \in V_\ell$; as described below.

Lemma 8 *For each level V_ℓ, each bag $s \in V_\ell$ and big entry/exit-bag-bucket configuration (s, b_i, x, z), w.h.p. $|X_i^{s,x,z}| \geq \frac{\epsilon^2}{\delta \log n} \cdot n_{s,i}^{x,z}$.*

Proof. Suppose (s, b_i, x, z) is a big entry/exit-bag-bucket configuration at some level V_ℓ. Let $p_1, \ldots, p_{n_{s,i}^{x,z}}$ be the partial tours in the entry/exit-bag-bucket configuration (s, b_i, x, z). Let the tour in OPT corresponding to p_i be $\mathcal{T}$. Two copies of tour p_i are assigned to b_i if both of the following events are true:

- Let A_i be the event where tour $\mathcal{T}$ is sampled as an extra tour. Since each tour is sampled with probability ϵ, we have $\mathbb{P}[A_i] = \epsilon$.

- Let B_i be the event where tour $\mathcal{T}$ is assigned to level ℓ. There are $h = \delta \log n$ many levels and since $\mathcal{T}$ (if sampled) is assigned to any one of its levels, $\mathbb{P}[B_i] \geq 1/h \geq 1/(\delta \log n)$.

Let Y_i be a random variable which is 1 if p_i is an extra tour in (v, b_i) and 0 otherwise.

$$\mathbb{E}[Y_i] = \mathbb{P}[Y_i = 1] = \mathbb{P}[A_i \wedge B_i] = \mathbb{P}[A_i] \cdot \mathbb{P}[B_i] \geq \epsilon/(\delta \log n).$$

Let $Y_i^{s,x,z} = \sum_{i=1}^{n_{s,i}^{x,z}} Y_i$ be the random variable keeping track of the number of sampled tours in (s, b_i, x, z). The number of extra tours, $|X_i^{s,x,z}| = 2Y_i^{s,x,z}$ since we add two copies of a sampled tour to $X_i^{s,x,z}$. By linearity of expectation, we have

$$\mathbb{E}[|X_i^{s,x,z}|] = 2\mathbb{E}[Y_i^{s,x,z}] = 2\sum_{i=1}^{n_{s,i}^{x,z}} \mathbb{E}[Y_i] \geq \frac{2\epsilon}{\delta \log n} \cdot n_{s,i}^{x,z}.$$

We want to show that $|X_i^{s,x,z}| \geq \frac{\mathbb{E}\left[|X_i^{s,x,z}|\right]}{2} \geq \frac{\epsilon}{\delta \log n} \cdot n_{s,i}^{x,z}$ with high probability over all vertex-bucket pairs.

Using Chernoff Bound with $\mu = \mathbb{E}[|X_i^{s,x,z}|] \geq \frac{2\epsilon^2}{\delta \log^2 n} \cdot n_{s,i}^{x,z} \geq 24 \log n$ since $n_{s,i}^{x,z} \geq \alpha \log^2 n/\epsilon$ and $\alpha \geq 20\delta$.

$$\mathbb{P}\left[|X_i^{s,x,z}| < \frac{\mathbb{E}[|X_i^{s,x,z}|]}{2}\right] \leq e^{-(5 \log n)} = \frac{1}{n^5}$$

Note that the above equation only shows the concentration bound for a single entry/exit-bag-bucket configuration. For a bag, there are $O(k^2)$ many entry/exit pairs. There are $O(kn)$ bags and $\tau = O(\log n/\epsilon)$ buckets, so the total number of entry/exit-bag-bucket configuration is at most $O(k^2 n \log n/\epsilon)$. Suppose we do a union bound over all buckets, we get

$$\sum_{\text{all } (s,b_i,x,z) \text{ configurations}} \mathbb{P}\left[|X_i^{s,x,z}| < \frac{\mathbb{E}[|X_i^{s,x,z}|]}{2}\right] \leq \frac{1}{n}.$$

We showed that for every entry/exit-bag-bucket configuration (s, b_i, x, z), $|X_i^{s,x,z}| \geq \frac{\epsilon}{\delta \log n} n_{s,i}^{x,z}$ holds with high probability. ∎

Lemma 9 *Consider all bags $s \in V_\ell$, big entry/exit-bag-bucket configuration (s, b_i, x, z) and the partial tours in $G_{i,g}^{s,x,z}$. We can modify the tours in $X_i^{s,x,z}$ (without increasing the cost) and adding some extra tokens at nodes in s (if needed) so that:*

1. *The tokens picked up by partial tours in $G_{i,g}^{s,x,z}$ are covered by some tour in $X_i^{s,x,z}$, and*

2. *The new partial tours that pick up the orphant tokens in $G_{i,g}^{s,x,z}$ have size exactly $h_{i,g}^{s,x,z,\max}$ and all tours still have size at most Q.*

3. *For each (new) partial tour of $X_i^{s,x,z}$ and every level $\ell' > \ell$, the size of partial tours of $X_i^{s,x,z}$ at a bag s' at level ℓ' is also one of $O((\log Q \log^2 n)/\epsilon^2)$ many possible sizes.*

Proof. Our proof is going to be very similar to Lemma 3 for the case of trees. Our goal is to use the extra tours in $X_i^{s,x,z}$ to cover tokens picked up by partial tours of $G_{i,g}^{s,x,z}$ and we want each extra tour in $X_i^{s,x,z}$ to cover exactly $h_{i,g}^{s,x,z,\max}$ tokens. The tours in the last group, $G_{i,g}^{s,x,z}$, cover $\sum_{t \in G_{i,g}^{s,x,z}} |t|$ many tokens. We will add $\sum_{t \in G_{i,g}^{s,x,z}} (h_{i,g}^{s,x,z,\max} - |t|)$ extra tokens in node x at bag s for each entry/exit-bag-bucket configuration (s, b_i, x, z) so that there are $h_{i,g}^{s,x,z,\max}$ tokens corresponding to each partial tour in $G_{i,g}^{s,x,z}$. From now on, we will assume each partial tour in a last group $G_{i,g}^{s,x,z}$ covers $h_{i,g}^{s,x,z,\max}$ tokens.

Using Lemma 8, we know with high probability that $|X_i^{s,x,z}|/|G_{i,g}^{s,x,z}| \geq 2$ since $|X_i^{s,x,z}| \geq \frac{\epsilon}{\delta \log n} \cdot n_{s,i}^{x,z} = 2|G_{i,g}^{s,x,z}|$. Let $Y_i^{s,x,z}$ denote the number of tours in entry/exit-bag-bucket configuration (s, b_i, x, z) that were sampled, so $|X_i^{s,x,z}| = 2|Y_i^{s,x,z}|$ and $|Y_i^{s,x,z}| \geq |G_{i,g}^{s,x,z}|$ with high probability. We will start by creating a one-to-one mapping $s : G_{i,g}^{s,x,z} \to Y_i^{s,x,z}$ which maps each tour in $G_{i,g}^{s,x,z}$ to a sampled tour in $Y_i^{s,x,z}$. We know such a one-to-one mapping exists since $|Y_i^{s,x,z}| \geq |G_{i,g}^{s,x,z}|$.

Let $\mathcal{T}$ be a sampled tour in $Y_i^{s,x,z}$ with two extra copies of it, $\mathcal{T}_1$ and $\mathcal{T}_2$ in $X_i^{s,x,z}$. Let the partial tours of $\mathcal{T}$ at the bottom part in V_ℓ be $p_1, \ldots, p_m$. We know $|\mathcal{T}| \geq \sum_{i=1}^{m} |p_i|$. Like the case for trees, s maps at most one tour in $G_{i,g}^{s,x,z}$ to each p_j. If a tour from $G_{i,g}^{s,x,z}$ maps to p_j, we will assume the load assigned to p_j would be $r_j = h_{i,g}^{s,x,z,\max}$ and p_j has load 0 if no tour is assigned to it.

Suppose we think of $r_1, \ldots, r_m$ as items and $\mathcal{T}_1$ and $\mathcal{T}_2$ as bins of size Q. We might not be able to fit all items $r_1, \ldots, r_m$ into a bin of size Q because

$\sum_{i=1}^{m} |r_i| \leq (1+\epsilon) \sum_{i=1}^{m} |p_i| \leq (1+\epsilon)|\mathcal{T}| \leq (1+\epsilon)Q$. Similar to the case of trees, we can show that we can assign $r_1, \ldots, r_j$ (for the maximum j) to $\mathcal{T}_1$ such that $\sum_{i=1}^{j} |r_i| \leq Q$ and the rest, $r_{j+1}, \ldots, r_m$ to $\mathcal{T}_2$ such that both $\mathcal{T}_1$ and $\mathcal{T}_2$ cover at most Q tokens and all items $r_1, \ldots, r_m$ are covered by either $\mathcal{T}_1$ or $\mathcal{T}_2$. Hence, we have shown that the extra partial tours pick up exactly $h_{i,g}^{s,x,z,\max}$ while picking up orphant tokens from $G_{i,g}^{s,x,z}$.

Also, the size of the extra tours after this modification at each bag s' at any level $\ell' > \ell$ is essentially the same as what each of r_i's were at those levels and since we go bottom to top in the tree, each of those partial tours r_i have a size that either belongs to a small bucket (and hence has one of $\alpha \log^2 n/\epsilon$ many sizes) or a big entry/exit-bag bucket (and hence has one of $O((\log Q \log n)/\epsilon^2)$ many sizes). Therefore, the size of partial tours of $X_i^{s,x,z}$ at any bag s' at level $\ell' > \ell$ is one of $O((\log Q \log^2 n)/\epsilon^2)$ many sizes. ■

Therefore, using Lemma 9, all the tokens of C_s remain covered by partial tours; those partial tours in $G_{i,j}^{s,x,z}$ (for $1 \leq j < g$) are tied to the top parts of the tours from group $G_{i,j+1}^{s,x,z}$ and the partial tours of $G_{i,g}^{s,x,z}$ will be tied to extra tours designated to level ℓ. We also add extra tokens at nodes in s to be picked up by the partial tours of C_s so that each partial tour has a size exactly equal to the maximum size of a group. All in all, the extra cost paid to build OPT_ℓ (from $\mathrm{OPT}_{\ell+1}$) is for the extra tours designated to level ℓ.

Theorem 8 ***(Structure Theorem)*** *Let* OPT *be the cost of the optimal solution to instance $\mathcal{I}$. We can build an instance $\mathcal{I}'$ such that each node has ≥ 1 tokens and there exists a near-optimal solution* OPT' *for $\mathcal{I}'$ having expected cost* $(1+2\epsilon)$OPT *with the following property. The partial tours going down C_s for every bag s in* OPT' *has one of $O((\log Q \log^2 n)/\epsilon^2)$ possible sizes. More specifically, suppose (s, b_i, x, z) is a entry/exit-bag-bucket configuration for* OPT'*. Then either:*

- *b_i is a small bucket and hence there are at most $\alpha \log^2 n/\epsilon$ many partial tours of C_s whose size is in bucket b_i, or*
- *b_i is a big bucket; in this case there are $g = (2\delta \log n)/\epsilon$ many group sizes*

in b_i: $\sigma_i \leq h_{i,1}^{s,x,z,max} \leq \ldots \leq h_{i,g}^{s,x,z,max} < \sigma_{i+1}$ and every tour of bucket i has one of these sizes.

Proof. We will show how to modify OPT to a near-optimal solution OPT′. We start from $\ell = h$ and let $\mathrm{OPT}_\ell = \mathrm{OPT}$. For decreasing values of ℓ we show, for each ℓ how to modify $\mathrm{OPT}_{\ell+1}$ to obtain OPT_ℓ. We do this in the following manner: we do not modify partial tours in small entry/exit-bag-bucket configuration. However, for tours in big entry/exit-bag-bucket configuration (s, b_i, x, z) in level $\ell - 1$, we place them into g groups $G_{i,1}^{s,x,z}, \ldots, G_{i,g}^{s,x,z}$ of equal sizes by placing the i'th $n_{s,i}^{x,z}/g$ partial tours into $G_{i,j}^{s,x,z}$. We have a mapping f from each partial tour in $G_{i,j-1}^{s,x,z}$ to one in $G_{i,j}^{s,x,z}$ for $j \in \{2, \ldots, g\}$. We modify OPT_ℓ to OPT_{l+1} in the following way: for each tour $\mathcal{T}$ that has a partial tour $t \in G_{i,j}^{s,x,z}$, replace the bottom part of $\mathcal{T}$ at s from t to $f(t)$ (which is in $G_{i,j-1}^{s,x,z}$). For each tour $t \in G_{i,j-1}^{s,x,z}$, we will add $h_{i,j-1}^{s,x,z,\max} - |t|$ many extra tokens at x in s. Note that by this change, the size of any tour such as $\mathcal{T}$ can only decrease and we are not violating feasibility of the tour because $h_{i,j-1}^{s,x,z,\max} \leq h_{i,j}^{s,x,z,\min}$. However, the tokens in C_s picked up by the partial tours in $G_{i,g}^{s,x,z}$ are not covered by any tours. We can use Lemma 9 to show how we can use extra tours to cover the partial tours in $G_{i,g}^{s,x,z}$ such that the new partial tours have size exactly $h_{i,g}^{s,x,z,\max}$.

We will inductively repeat this for levels $\ell - 2, \ell - 3, \ldots, 1$ and obtain $\mathrm{OPT}_1 = \mathrm{OPT}'$. Note that by adding extra tokens $h_{i,j-1}^{s,x,z,\max} - |t|$ for a tour $t \in G_{i,j-1}^{s,x,z}$, we are enforcing that the coverage of each tour is the maximum size of tours in its group. In a big bucket, there are $g = (2\delta \log n)/\epsilon$ many group sizes, so there are $O(\log n/\epsilon)$ possible sizes for tours in big entry/exit-bag-bucket configuration at a node. In a small entry/exit-bag-bucket configuration, there can be at most $\alpha \log^2 n/\epsilon$ many tours and since there are $\tau = O(\log Q/\epsilon)$ many buckets, there can be at most $O((\log Q \log^2 n)/\epsilon^2)$ many tour sizes covering C_b.

Using Lemma 7, we know the expected cost of the extra tours is at most $2\epsilon \cdot \textsc{opt}$, so the expected cost of $\textsc{opt}' \leq (1 + 2\epsilon)\textsc{opt}$. ∎

3.3 Dynamic Program

In this section we prove Theorem 3 by presenting a dynamic program that will compute a near optimum solution guaranteed by the structure theorem (Theorem 8). For a given bag s, we will estimate the number of tours entering and exiting s. Informally, we will have a vector $\vec{n}^{s,x,z} \in [n]^{\tau}$ where if $i < 1/\epsilon$, $\vec{n}_i^{s,x,z}$ keeps track of the exact number of tours covering i tokens in C_s by entering through x and exiting though z and if $i \geq 1/\epsilon$, $\vec{n}_i^{s,x,z}$ keeps track of the number of tours covering between $[\sigma_i, \sigma_{i+1})$ tokens. Let a_s denote the total number of tokens to be picked up from nodes from bags below and including bag s. Since each bag s has k nodes, we use $\vec{o}_s \in [n]^k$ to denote the extra tokens to be picked up from nodes at bag s. If v is a node in bag s, then $\vec{o}_{s,v}$ denotes the number of extra tokens to be picked up at v in s. For a given entry/exit-bag-bucket configuration (s, b_i, x, z), we will keep track of other pieces of information conditional on whether it is small or big. If entry/exit-bag-bucket configuration (s, b_i, x, z) is small, we will store all tour sizes exactly. Since the number of tours in a small entry/exit-bag-bucket configuration is at most $\gamma = \alpha \log^2 n/\epsilon$, we will use a vector $\vec{t}^{s,x,z,i} \in [n]^{\gamma}$ to represent the tours where $\vec{t}_j^{s,x,z,i}$ represents the size of the j-th tour in the i-th bucket of tours covering C_s entering through x and exiting through z.

If the entry/exit-bag-bucket configuration (s, b_i, x, z) is big, there are $g = (2\delta \log n)/\epsilon$ many tour sizes corresponding to $n^{O(g)}$ possibilities. For each entry/exit-bag-bucket configuration (s, b_i, x, z), we need to keep track of the following information,

- $\vec{h}^{s,x,z,i} \in [n]^g$ is a vector where $\vec{h}_j^{s,x,z,i} = h_{i,j}^{s,x,z,\max}$, which is the size of the maximum tour which lies in group $G_{i,j}^{s,x,z}$ of bucket i at bag s entering through x and exiting through z.
- $\vec{l}^{s,x,z,i} \in [n]^g$ is a vector where $\vec{l}_j^{s,x,z,i}$ denotes the number of partial tours covering $h_{i,j}^{s,x,z,\max}$ tokens which lies in group $G_{i,j}^{s,x,z}$ of bucket i at bag s entering through x and exiting through z.

For a bag s and entry/exit pairs, let $\vec{p}_{s,x,z}$ be a vector containing information

about all tours entering and exiting s through x and z across all buckets.

$$\vec{p}_{s,x,z} = [\vec{n}^{s,x,z}, (\vec{t}^{s,x,z,1}, \vec{h}^{s,x,z,1}, \vec{l}^{s,x,z,1}), (\vec{t}^{s,x,z,2}, \vec{h}^{s,x,z,2}, \vec{l}^{s,x,z,2}), \ldots, (\vec{t}^{s,x,z,\tau}, \vec{h}^{s,x,z,\tau}, \vec{l}^{s,x,z,\tau})].$$

Similar to the case of trees, an entry/exit-bag-bucket configuration (s, b_i, x, z) is either small or big and cannot be both, hence given $(\vec{t}^{s,x,z,i}, \vec{h}^{s,x,z,i}, \vec{l}^{s,x,z,i})$, it cannot be the case that $\vec{t}^{s,x,z,i} \neq \vec{0}, \vec{h}^{s,x,z,i} \neq \vec{0}$ and $\vec{l}^{s,x,z,i} \neq \vec{0}$. Since a bag s contains $O(k)$ nodes, then we will let $\vec{y}_s$ denote a configuration of all partial tours covering tokens in C_s which are entering and exiting s. Let $v_1, \ldots, v_d$ be the set of all nodes in s, then $\vec{y}_s$ contains information of tours entering and exiting s through pairs of nodes in $\{v_1, \ldots, v_d\}$. Note that a tour can enter and exit s through the same node.

$$\vec{y}_s = [a_s, \vec{o}_s, \vec{p}_{s,v_1,v_1}, \vec{p}_{s,v_1,v_2}, \ldots, \vec{p}_{s,v_d,v_{d-1}}, \vec{p}_{s,v_d,v_d}].$$

The subproblem $\mathbf{A}[s, \vec{y}_s]$ is supposed to be the minimum cost collection of partial tours covering C_s having tour profiles corresponding to $\vec{y}_s$. Our dynamic program heavily relies on the properties of the near-optimal solution characterized by the structure theorem. We will compute $\mathbf{A}[\cdot, \cdot]$ in a bottom-up manner, computing $\mathbf{A}[s, \vec{y}_s]$ after we have computed entries for the children bags of s.

The final answer is obtained by looking at various entries of the root bag of the tree decomposition, denoted by r_s. We will take the minimum cost entry amongst $\mathbf{A}[r_s, \vec{y}_{r_s}]$ such that $\vec{y}_{r_s}$ is the configuration where all tours enter and exit r_s only through the depot, r. We will compute our solution in a bottom-up manner.

For any nodes u, v in bag s, if there is no edge between u and v, we can add an edge between them and the cost of the edge is the shortest path cost between u and v in G. Similarly, for two adjacent bags, s and s_1, if $u \in s$ and $v \in s_1$ and if there is no edge between u and v in G, we will add an edge between them and the cost of the edge is the shortest path cost between u and v in G. If $u = v$, then the cost of the edge connecting them can be assumed to be zero. Let $\|\vec{o}_s\| = \sum_{u \in s} \vec{o}_{s,u}$.

For the base case, we consider leaf bags. A leaf bag s could have $a_s \geq 1$ tokens where $a_s = \|\vec{o}_s\|$. We will defer how we compute $\mathbf{A}[s, \vec{y}_s]$ to the end

of this section. Informally, we will set $\mathbf{A}[s, \vec{y}_s]$ to be the minimum cost of the edges between nodes in bag s used for the tours in $\vec{y}_s$ to pick up $\vec{o}_s$ tokens located at nodes in bag s. The total capacity of the tours in $\vec{y}_s$ should be exactly a_s and a token at a node should be picked up by one of the tours in $\vec{y}_s$. From our structure theorem, we know there exists a near optimum solution such that each partial tour has one of $O(\log Q \log^2 n/\epsilon^2)$ tour sizes and for each small bucket, there are at most $\alpha \log^2 n/\epsilon$ partial tours in it. For every big bucket, there are $g = (2\delta \log n)/\epsilon$ many group sizes and every tour of bucket i has one of those sizes. We are computing all possible $\mathbf{A}[s, \vec{y}_s]$ entries and from our structure theorem, we know one of them has near-optimum expected cost, so by enumerating all possibilities, our dynamic program finds a near-optimums solution for the leaf bag, proving the base case.

Recall that the tree T is binary. Suppose bag s has two children in T, s_1 and s_2. To compute cell $\mathbf{A}[s, \vec{y}_s]$, we will use the entries of its children, $\mathbf{A}[s_1, \vec{y}']$ and $\mathbf{A}[s_2, \vec{y}'']$. Suppose C_{s_i} has a_{s_i} tokens, then $a_s = \|\vec{o}_s\| + a_{s_1} + a_{s_2}$. $\mathbf{H}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']$ checks whether the tour profiles $\vec{y}_s, \vec{y}'$ and $\vec{y}''$ are consistent meaning that all tokens picked up by tours in $\vec{y}'$ and $\vec{y}''$ along with tokens in s, $\vec{o}_s$ are picked up by tours in $\vec{y}_s$. We will also define $\mathbf{I}[\cdot, \cdot, \cdot, \cdot]$ where $\mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']$ denotes the cost of using the edges in bag s, edges connecting nodes in s and s_1, and edges connecting nodes in s and s_2. We can think of $\mathbf{I}$ as the cost of using edges to patch up partial tours covering C_{s_1} and partial tours covering C_{s_2} to create tours covering C_s. We will explain in the next section how $\mathbf{H}$ and $\mathbf{I}$ are computed. Recall $\vec{o}_s$ is part of $\vec{y}_s$. Suppose we have already computed the entries $\mathbf{A}[s_1, \cdot]$ and $\mathbf{A}[s_2, \cdot]$, we will compute $\mathbf{A}[s, \cdot]$ in the following way:

$$\mathbf{A}[s, \vec{y}_s] = \min_{\vec{y}', \vec{y}'' : \mathbf{H}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}''] = \text{True}} \{\mathbf{A}[s_1, \vec{y}'] + \mathbf{A}[s_2, \vec{y}''] + \mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']\}.$$

There are four possibilities for each partial tour t at bag s going down C_s covering tokens for the subtree rooted at children bags, s_1 and s_2 while also picking up extra tokens from nodes in s:

- t could be a tour that picks up tokens from nodes at bag s and does not visit or pick up tokens in $C_{s_1} \cup C_{s_2}$.

- t could be a tour that picks up tokens from nodes at bag s and picks up tokens only from C_{s_1}.
- t could be a tour that picks up tokens from nodes at bag s and picks up tokens only from C_{s_2}.
- t could be a tour that picks up tokens from nodes at bag s and picks up tokens from $C_{s_1} \cup C_{s_2}$.

We would find the minimum cost over all configurations $\vec{y}_s, \vec{y}', \vec{y}''$ as long as $\vec{y}_s, \vec{y}', \vec{y}''$ are consistent. We say $\vec{y}_s, \vec{y}', \vec{y}''$ are consistent if there is a way to write each tour in $\vec{y}_s$ as a combination of at most one tour from $\vec{y}'$, at most one tour from $\vec{y}''$ while also picking up extra tokens from nodes in s. We would also require that all tokens in $\vec{y}'$ and $\vec{y}''$ are picked up by tours in $\vec{y}_s$.

For a leaf bag s, $\mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{0}, \vec{0}]$ denotes the minimum cost of tours entering bag s and visiting the nodes in s such that all tokens in s are picked up by some tour in $\vec{y}_s$. The last two entries are set to $\vec{0}$ since s is a leaf bag, and has no children, and there are no other tours (apart from those in $\vec{y}_s$) entering or exiting through nodes in bag s. We will set $\mathbf{A}[s, \vec{y}_s] = \mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{0}, \vec{0}]$ since $\mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{0}, \vec{0}]$ computes exactly the minimum cost collection of partial tours covering $C_s = s$ having tour profiles corresponding to $\vec{y}_s$. We will explain how to compute the entries of $\mathbf{I}[\cdot, \cdot, \cdot, \cdot]$ in the next section.

3.3.1 Checking Consistency

In our dynamic program, we are given three vectors $\vec{y}_s, \vec{y}', \vec{y}''$ where s is a bag having child bags s_1 and s_2. $\vec{y}'$ represents the configuration of tours covering C_{s_1} and $\vec{y}''$ represents the configuration of tours covering C_{s_2}. Given a $\vec{y}_s$, for each node u in s, there are $\vec{o}_{s,u}$ many tokens to be picked up at u. We require the tokens for nodes in s and tokens covered by the partial tours from $\vec{y}'$ and $\vec{y}''$ to be picked up by tours in $\vec{y}_s$. For simplicity, we will refer to a tour from $\vec{y}_s$ as t_s, $\vec{y}'$ as t_u and a tour from $\vec{y}''$ as t_w.

Definition 9 *We say configurations $\vec{y}_s, \vec{y}'$ and $\vec{y}''$ are **consistent** if the following holds:*

- *Every tour in $\vec{y}'$ maps to some tour in $\vec{y}_s$.*
- *Every tour in $\vec{y}''$ maps to some tour in $\vec{y}_s$.*
- *Every tour in $\vec{y}_s$ has at most two mapping to it and both cannot be from $\vec{y}'$ or $\vec{y}''$.*
- *Suppose only one tour t_u (t_w) maps to a tour t_s in $\vec{y}_s$. The number of extra tokens (from nodes in s) in total picked up by tour t_s from nodes in bag s is exactly $|t_s| - |t_u|$ ($|t_s| - |t_w|$).*
- *Suppose t_s has two tours: t_u in $\vec{y}'$ and t_w in $\vec{y}''$ mapping to it, then the number of extra tokens (from nodes in s) picked up by tour t_s at s is exactly $|t_s| - |t_u| - |t_w|$.*
- *All tokens of nodes at bag s, $\vec{o}_s$ are picked up tours in $\vec{y}_s$.*

Consistency ensures that we can patch up tours from subproblems and combine them into new tours in a correct manner while also picking up extra tokens from nodes in s. We will describe how we can compute consistency. Instead of using $\vec{y}_s$, we will use $\vec{z}_s$ which is the same as $\vec{y}_s$, but excludes information about the number of tokens in a bag, and only tracks information about the number of tours passing through bag s.

$$\vec{z}_s = [\vec{p}_{s,v_1,v_1}, \vec{p}_{s,v_1,v_2}, \ldots, \vec{p}_{s,v_d,v_{d-1}}, \vec{p}_{s,v_d,v_d}].$$

We will similarly define $\vec{z}'$ and $\vec{z}''$. Suppose t_{s,x_1,x_2} is a tour in s which enters through x_1 and exits through x_2, let $\vec{z}_s - t_{s,x_1,x_2}$ refers to the configuration $\vec{z}_s$ having one less tour of size $|t_{s,x_1,x_2}|$ from tours entering through x_1 and exiting through x_2 in s. Recall that $\vec{o}_s$ is the vector of extra tokens at each node in bag s which need to be covered by tours in $\vec{z}_s$.

Given $\vec{z}_s, \vec{z}', \vec{z}''$ and $\vec{o}_s$, we will use the table $\mathbf{H}$ to check if $\vec{z}_s, \vec{z}', \vec{z}''$ are consistent. Let $\mathbf{H}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ =True if $\vec{z}_s, \vec{z}'$ and $\vec{z}''$ are consistent and False otherwise. For the base case, $\mathbf{H}[\vec{0}, \vec{0}, \vec{0}, \vec{0}]$ =True. For the recurrence, we will look at all possible ways of combining tours from $\vec{z}'$ and $\vec{z}''$ into $\vec{z}_s$ while also picking up extra tokens from bag s. For a tour t_s, let $\vec{o}'_{s,t_s}$ be a vector where

$\vec{o}'_{s,t_s,u}$ denotes the number of extra tokens picked up by t_s at node u in bag s. Let $\left\|\vec{o}'_{s,t_s}\right\| = \sum_{u \in s} \vec{o}'_{s,t_s,u}$ count the number of tokens picked up by t_s from nodes in s.

Recall that a tour t_s merges with at most one tour t_u from $\vec{z}'$ and at most one tour t_w from $\vec{z}''$. Similar to the case of trees, we can write the recurrence of our consistency table as:

$$\mathbf{H}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}''] = \bigvee_{\substack{t_s, t_u, t_w, \vec{o}'_{s,t_s} \\ |t_s| = |t_u| + |t_w| + \left\|\vec{o}'_{s,t_s}\right\|}} \mathbf{H}[\vec{o}_s - \vec{o}'_{s,t_s}, \vec{z}_s - t_s, \vec{z}' - t_u, \vec{z}'' - t_w].$$

Although the above DP lets us check if $\vec{y}_s, \vec{y}'$ and $\vec{y}''$ are consistent, the entries of $\mathbf{H}$ are True/False and does not give us information about the optimum order in which tour t_s should visit nodes in s or the cost associated with such an ordering. Suppose the tour t_s visited k_s nodes in bag s, there are $O(k^{k_s})$ many paths that tour t_s can choose to take and each path has a cost associated with it. Our goal is to find a path having the smallest cost while also picking up tokens from nodes in bag s. We will next compute the minimum cost way to visit nodes in s and pick up tokens from them. Recall the recurrence of our dynamic program for $\mathbf{A}$ is the following,

$$\mathbf{A}[s, \vec{y}_s] = \min_{\vec{y}', \vec{y}'' : \mathbf{H}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}''] = \text{True}} \{\mathbf{A}[s_1, \vec{y}'] + \mathbf{A}[s_2, \vec{y}''] + \mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']\}.$$

The cost of using edges in C_{s_1} and C_{s_2} by the partial tours in $\vec{y}'$ and $\vec{y}''$ in $\vec{y}_s$ are accounted for by $\mathbf{A}[s_1, \vec{y}'] + \mathbf{A}[s_2, \vec{y}'']$. However, we have not accounted for the cost of hopping from one node to the other in s and also the cost of going from nodes in s to nodes in child bags, s_1 and s_2. Note that a tour t_s enters and exits through each node in s at most once. Note that a tour visits a node u in s if it either has to pick up tokens at u or if it uses u to enter the child bag. If a tour t_s enters and exist a node two or more times, we can short cut it so that it enters and exits only once. A tour in t_s can visit up to k nodes in a bag s and it could use one of the nodes to enter a child bag (s_1 or s_2) and if so, it would use a node in s to return to the bag s. This means the tour t_s could visit up to k nodes in s. Let P_{t_s} be the ordered collection of edges where either both endpoints are in s or one endpoint is in s and the other is in

$s_1 \cup s_2$. There are $O((3k)^{3k})$ possible permutations of for P_{t_s} and t_s could pick up at most Q tokens from each node that it visits and each permutation has an associated cost with it. The number of possibilities for P_{t_s} characterized by the the order of visiting nodes and the number of tokens picked up by tour t_s from the k nodes in bag s is at most $O(Q^k(3k)^{3k})$. We will let cost(P_{t_s}) denote the cost of the edges in P_{t_s}. The following figure illustrates an example of one such tour t_s (in red) and P_{t_s} (in blue).

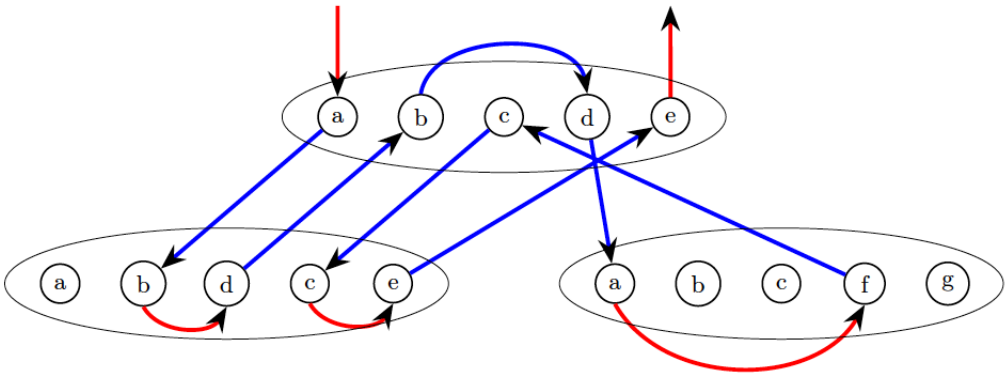


Figure 3.1: Blue edges represent one such edge set for a particular tour t_s

Although $\mathbf{H}$ tells us if $\vec{y}_s, \vec{y}'$ and $\vec{y}''$ are consistent, $\mathbf{H}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ does not give us the cost of patching up $\vec{y}'$ and $\vec{y}''$ to form $\vec{y}_s$. We will use $\mathbf{H}$ to compute $\mathbf{I}$. Let $\mathbf{I}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ denote the cost of using the edges in bag s, edges connecting nodes in s and s_1, and edges connecting nodes in s and s_2. We can think of $\mathbf{I}$ as the cost of using edges to patch up partial tours covering C_{s_1}, $\vec{z}'$ and partial tours covering C_{s_2}, $\vec{z}''$, to create tours covering C_s, $\vec{z}_s$. For the base case, we will set $\mathbf{I}[\vec{0}, \vec{0}, \vec{0}, \vec{0}] = 0$ and set all other entries to infinity. We will only compute an entry $\mathbf{I}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ if $\mathbf{H}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ =True. Along with all possible values of $\vec{o}'_s, t_s, t_u, t_w$, we will also look at all possible paths P_{t_s}. In our recurrence, we are taking a tour t_s from $\vec{y}_s$ along with maybe a tour t_u from $\vec{y}'$, maybe a tour t_w from $\vec{y}''$ along with tokens $\vec{o}'_s$ that t_s covers at nodes in bag s. For such a tour t_s, there are $O(Q^k(3k)^{3k})$ many possibilities for P_{t_s}. For a fixed P_{t_s}, cost(P_{t_s}) is the cost of forming t_s from patching up t_u and t_w while picking up extra tokens from nodes in s. We will enumerate through

all possibilities, break the recurrence into subproblems and find a solution of minimum cost. We can write the recurrence as follows:

$$\mathbf{I}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}''] = \min_{\substack{t_s, t_u, t_w, P_{t_s}, \vec{o}'_{s,t_s} \\ |t_s|=|t_u|+|t_w|+\left\|\vec{o}'_{s,t_s}\right\|}} \left\{\text{cost}(P_{t_s}) + \mathbf{I}[\vec{o}_s - \vec{o}'_{s,t_s}, \vec{z}_s - t_s, \vec{z}' - t_u, \vec{z}'' - t_w]\right\}.$$

3.4 Extension to Splittable CVRP

We will extend our algorithm for unit demand CVRP on bounded-treewidth graphs to the splittable CVRP when demands are quasi-polynomially bounded. In our algorithm for unit demand CVRP for bounded-treewidth CVRP, we viewed the unit demand of each node as a token placed at the node. For the splittable case, we can rescale the demand $d(v)$ such that there are $1 \leq d(v) < nQ$ tokens on a node and we can use the same structure theorem as before by modifying tours such that there are at most $O(\log Q \log^2 n/\epsilon^2)$ different tours for partial tours at a node. We can use the same DP to compute the solution. Each $\vec{y}_s$ consists of $O(k^2)$ different $\vec{p}_{s,u,v}$ vectors. Each $\vec{p}_{s,u,v}$ contains τ many triples $(\vec{t}^{s,x,z,i}, \vec{h}^{s,x,z,i}, \vec{l}^{s,x,z,i})$.

1. Each $\vec{t}^{s,x,z,i}$ has $(nQ)^{O(\log^2 n/\epsilon^2)}$ possibilities since there are at most $O(\log^2 n/\epsilon)$ tours in a small bucket.

2. Each $\vec{h}^{s,x,z,i}$ and $\vec{l}^{s,x,z,i}$ have $(nQ)^{O(g)}$ possibilities. Recall that $g = (2\delta \log n)/\epsilon^2$, so each $\vec{h}^{s,x,z,i}$ and $\vec{l}^{s,x,z,i}$ have $(nQ)^{O(\log n/\epsilon^2)}$ possibilities.

3. Each triple $(\vec{t}^{s,x,z,i}, \vec{h}^{s,x,z,i}, \vec{l}^{s,x,z,i})$ has $(nQ)^{O(\log^2 n/\epsilon)}$ possibilities.

4. Since $\vec{p}_{s,u,v}$ has $\tau = O(\log Q/\epsilon)$ many such triples, the number of possible entries for $\vec{p}_{s,u,v}$ is $(nQ)^{O(\tau \log^2 n/\epsilon)} = (nQ)^{O(\log Q \log^2 n/\epsilon^2)}$.

5. Since $\vec{y}_s$ consists of $O(k^2)$ different entries of $\vec{p}$, the total number of possible entries for each $\vec{y}_s$ is $(nQ)^{O(k^2 \log Q \log^2 n/\epsilon^2)}$.

Similar to the analysis of the runtime of the unit demand case, the time complexity of computing the entries of DP tables $\mathbf{A}$ and consistency table $\mathbf{I}$ is, $(kQ)^{O(k)}(nQ)^{O(k^2 \log Q \log^2 n/\epsilon^2)} = (nQ)^{O(k^2 \log Q \log^2 n/\epsilon^2)}$ since $k \leq n$. Suppose $Q = n^{O(\log^c n)}$, then the runtime of our algorithm is $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$.

Chapter 4

Extension to Other Graphs Metrics

In this section, we will show how we can use our algorithm for CVRP on bounded-treewidth graphs as a blackbox to obtain a QPTAS for graphs of bounded doubling metrics and graphs of bounded highway dimension. Before diving into approximation schemes for graphs of bounded doubling or highway dimension, we will first proving a lemma about embedding metric spaces from one to another and its relation to CVRP.

4.1 Embedding Lemma for CVRP

Suppose G is the input graph in an instance for CVRP. Suppose we have a probabilsitic embedding $\phi : G \to H$ with distortion $(1 + \epsilon)$ i.e. for any two nodes u, v, on expectation

$$d_G(u, v) \leq d_H(u, v) \leq (1 + \epsilon)d_G(u, v).$$

We will refer to H as the host graph. We will show that if we have access to an algorithm to obtain a near-optimum solution in H, we can embed graph G into a host graph H, obtain a solution for instance H and lift the solution back to a solution for G at an ϵ fraction extra cost. We will state this as a lemma,

Lemma 10 *For any $\epsilon > 0$, let $\phi : G \to H$ be a probabilistic embedding with expected distortion $(1+\epsilon)$. For an instance G of Splittable CVRP with optimal*

solution OPT_G*, by embedding* G *into* H *using* ϕ *and lifting the solution* OPT_H *for the host graph* H *back to* G*, we can obtain a solution for* G *of cost at most* $(1+\epsilon)\mathrm{OPT}_G$.

Proof. Suppose $\mathcal{T}_1, \ldots, \mathcal{T}_c$ are the set of tours in OPT_G, let $E(\mathcal{T}_i)$ be the edges denoting the order in which nodes are visited by $\mathcal{T}_i$. Let $\mathrm{cost}_G(\mathcal{T}_i)$ be the contribution of the tour $\mathcal{T}_i$ towards OPT in G. We can write $\mathrm{cost}_G(\mathcal{T}_i) = \sum_{uv \in E(\mathcal{T}_i)} d_G(u,v)$ and we can write $\mathrm{OPT}_G = \sum_{i=1}^{c} \mathrm{cost}_G(\mathcal{T}_i) = \sum_{i=1}^{c} \sum_{uv \in E(\mathcal{T}_i)} d_G(u,v)$.

Let $\mathrm{cost}_G(\mathrm{OPT}_H)$ be the cost of using solution OPT_H to solve the instance on graph G. Our goal is to show that $\mathrm{cost}_G(\mathrm{OPT}_H) \leq (1+\epsilon)\mathrm{cost}_G(\mathrm{OPT}_G) \leq (1+\epsilon)\mathrm{OPT}_G$. Using the embedding ϕ, we know that for any two nodes u, v, on expectation,

$$d_G(u,v) \leq d_H(u,v) \leq (1+\epsilon) d_G(u,v)$$

Note that the vertices are the same in both H and G, so a solution to an instance H is a solution in G (and vice versa). Since $d_G(u,v) \leq d_H(u,v)$, we know for any solution OPT, $\mathrm{cost}_G(OPT) \leq \mathrm{cost}_H(\mathrm{OPT})$. Note that $\mathrm{cost}_H(\mathrm{OPT}_H) \leq \mathrm{cost}_H(\mathrm{OPT}_G)$.

$$\begin{aligned}
\mathrm{cost}_G(\mathrm{OPT}_H) &\leq \mathrm{cost}_H(\mathrm{OPT}_H) \leq \mathrm{cost}_H(\mathrm{OPT}_G) \\
&= \sum_{\mathcal{T}_i \in \mathrm{OPT}_G} \mathrm{cost}_H(\mathcal{T}_i) = \sum_{\mathcal{T}_i \in \mathrm{OPT}_G} \sum_{uv \in E(\mathcal{T}_i)} d_H(u,v) \\
&\leq \sum_{\mathcal{T}_i \in \mathrm{OPT}_G} \sum_{uv \in E(\mathcal{T}_i)} (1+\epsilon) d_G(u,v) \\
&= (1+\epsilon) \sum_{\mathcal{T}_i \in \mathrm{OPT}_G} \mathrm{cost}_G(\mathcal{T}_i) \\
&= (1+\epsilon)\mathrm{cost}_G(\mathrm{OPT}_G) = (1+\epsilon)\mathrm{OPT}_G
\end{aligned}$$

■

4.2 QPTAS for Graphs of Bounded Doubling Dimension

In this section, we will prove quasi-polynomial time approximation schemes for Splittable CVRP in graphs of bounded treewidth when $Q = n^{O(\log^c n)}$. We

will use the following result about emdedding graphs of doubling dimension D into a bounded-treewidth graph of treewidth $k \leq 2^{O(D)}\lceil\left(\frac{4D\log\Delta}{\epsilon}\right)^D\rceil$ by Talwar [28].

Lemma 11 *(Theorem 9 in [28]) Let (X,d) be a metric with doubling dimension D and aspect ratio Δ. For any $\epsilon > 0$, (X,d) can be $(1+\epsilon)$ probabilistically approximated by a family of treewidth k-metrics for $k \leq 2^{O(D)}\lceil\left(\frac{4D\log\Delta}{\epsilon}\right)^D\rceil$.*

The input graph G is embedded into a host graph H of low bounded treewidth using the embedding given in Lemma 11. The algorithm then finds a $(1+\epsilon)$-approximation for CVRP for H, using the dynamic programming solution from Chapter 3 for graphs of bounded treewidth. The solution for H is then *lifted* back to a solution in G. For each tour in the solution for H, a tour in G will visit nodes in the same order as the tour in H. The embedding also ensures that H has treewidth small enough that the algorithm runs in quasi-polynomial time. We will prove the following result and we have restated it for convenience,

Theorem 4 *For any $\epsilon > 0$ and $D > 0$, there is a an algorithm that, given an instance of the splittable CVRP with capacity $Q = n^{\log^c n}$ and the graph has doubling dimension D with cost* OPT*, finds a $(1+\epsilon)$-approximate solution in time $n^{O(D^D \log^{2c+D+3} n/\epsilon^{D+2})}$.*

Proof. This follows easily from Lemma 11, Lemma 10 and using the algorithm for bounded-treewidth as a blackbox. In place of k, we will substitute $k = 2^{O(D)}\lceil\left(\frac{4D\log\Delta}{\epsilon}\right)^D\rceil$ into the runtime for the algorithm for bounded-treewidth which is $n^{O(k^2\log^{2c+3} n/\epsilon^2)}$. Hence, we have an algorithm for graphs of bounded doubling dimension with runtime $n^{O(D^D \log^{2c+D+3} n/\epsilon^{D+2})}$. ∎

As an immediate corollary, since $\mathbb{R}^2$ has doubling dimension $\log_2 7$ [30], the above theorem implies an approximation scheme for unit demand CVRP on Euclidean metrics on $\mathbb{R}^2$ in time $n^{O(\log^6 n/\epsilon^6)}$ which improves on the run time of $n^{\log^{O(1/\epsilon)} n}$ of [17].

4.3 QPTAS for Graphs of Bounded Highway Dimension

In this section, we will prove quasi-polynomial time approximation schemes for Splittable CVRP in graphs of bounded highway dimension when $Q = n^{O(\log^c n)}$. We will use the following result by Feldmann et al. [18] related to graphs of low highway dimension.

Lemma 12 *(Theorem 3 in [18]) Let G be a graph with highway dimension D of violation $\lambda > 0$, and aspect ratio Δ. For any $\epsilon > 0$, there is a polynomial-time computable probabilistic embedding H of G with treewidth $(\log \Delta)^{O\left(\log^2(\frac{D}{\epsilon\lambda})/\lambda\right)}$ and expected distortion $1 + \epsilon$.*

The input graph G is embedded into a host graph H of low bounded treewidth using the embedding given in Lemma 12. The algorithm then finds a $(1+\epsilon)$-approximation for CVRP for H, using the dynamic programming solution from Chapter 3 for graphs of bounded treewidth. The solution for H is then *lifted* back to a solution in G. For each tour in the solution for H, a tour in G will visit nodes in the same order as the tour in H. The embedding also ensures that H has treewidth small enough that the algorithm runs in quasi-polynomial time. We will prove the following result and we have restated it for convenience,

Theorem 5 *For any $\epsilon > 0, \lambda > 0$ and $D > 0$, there is a an algorithm that, given an instance of the splittable CVRP with capacity $Q = n^{\log^c n}$ and if the graph has highway dimension D with violation λ with an optimla solution of cost* OPT*, finds a solution whose cost is at most $(1+\epsilon)$*OPT *in time $n^{O(\log^{2c+3+\log^2(\frac{D}{\epsilon\lambda})\cdot\frac{1}{\lambda}} n/\epsilon^2)}$.*

Proof. This follows easily from Lemma 12, Lemma 10 and using the algorithm for bounded-treewidth as a blackbox. In place of k, we will substitute $k = (\log \Delta)^{O\left(\log^2(\frac{D}{\epsilon\lambda})/\lambda\right)}$ into the runtime for the algorithm for bounded-treewidth which is $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$. Hence, we have an algorithm for graphs of bounded doubling dimension with runtime $n^{O(\log^{2c+3+\log^2(\frac{D}{\epsilon\lambda})\cdot\frac{1}{\lambda}} n/\epsilon^2)}$. ∎

Chapter 5

Conclusion and Open Problems

In this thesis, we presented QPTAS's for CVRP on trees, graphs of bounded treewidths, bounded doubling dimension, and bounded highway dimension. The immediate questions to consider are whether these approximation schemes can in fact be turned into PTAS's. Even for the case of trees, although we can improve the run time slightly by shaving off one (or maybe two) log factors from the exponent, it is not clear if it can be turned into a PTAS without significant new ideas. We will list some concrete open problems related to CVRP.

1. For the case of trees, in polynomial time, the best approximation ratio is 4/3. A natural open problem is whether one can obtain a PTAS for the case of trees.

2. A relaxed version of the above problem is also open: Recall that for an instance of CVRP on trees, we can solve it exactly in time $n^{O(Q)}$. Is it possible to obtain a polynomial-time approximation scheme for moderately large values of Q, say $Q = O(\log n)$?

3. Is there a PTAS for CVRP parameterized by the height of the tree? We proved a height reduction lemma for the case of trees which shows for a given tree instance T for CVRP with optimal cost OPT, we can obtain an instance T' with height $\delta \log^2 / \epsilon$ for some constant $\delta > 0$ such that a solution for T' lifted back to T has cost at most $(1+\epsilon)$OPT. Does there exist a similar height reduction scheme that reduces the height to

$O(\log n/\epsilon)$?

4. For graphs of bounded doubling or highway dimension, we used the results of Talwar [28] and Feldmann et al. [18] to embed the input graph into a graph of treewidth $O(\log^c n)$ for some constant c dependent on the highway or doubling dimension. There are two open problems related to this:

   (a) The current embedding is probabilistic and the distortion achieved only holds in expectation. Is it possible to obtain a deterministic embedding?

   (b) If one is able to achieve a PTAS for bounded-treewidth graphs, one can still only obtain QTPAS's using the embeddings from [28] and [18] since the treewidth is $\Omega(\log n)$. Is it possible to embed a graph of bounded highway or doubling dimension into graphs of $O(1)$ treewidth? Becker et al. [8] showed an embedding from graphs of bounded highway dimension to graphs of $O(1)$ treewidth when Q was fixed. Is there such an embedding when $Q = O(n)$?

5. For the Euclidean ($\mathbb{R}^2$) case, Adamaszek et al. [3] used the QPTAS for Euclidean by Das and Mathieu [17] as a black-box to obtain a PTAS for Euclidean for moderately large values of $Q \leq 2^{\log^\delta n}$, where $\delta = \delta(\epsilon)$. Can one use similar techniques to the work of Adamaszek et al. [3] and use our QPTAS for doubling or highway dimension as a black-box to obtain a PTAS for doubling or highway dimension for moderately large values of Q?

6. Becker and Paul [10] showed a bicriteria PTAS for the case of CVRP in trees. An open problem is to design a bicriteria PTAS for the graphs of bounded treewidth.

7. A major open is to get a PTAS for Euclidean ($\mathbb{R}^2$). As discussed in Adamaszek et al. [3], the difficult case appears to be when Q is polynomial in n (e.g. $Q = \sqrt{n}$).

8. Another interesting question is to consider CVRP on planar graphs and develop approximation schemes for them and, more generally graphs of bounded genus or minor free graphs.

9. For the splittable case, our algorithm only worked in quasi-polynomial time when $Q = n^{O(\log^c n)}$. Similar to how we rounded and rescaled edge weights/costs to make them polynomially bounded in n, is it possible to rescale the demands at a node to make the demand polynomially bounded in n at a small cost of $\epsilon \cdot \text{OPT}$?

References

[1] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck, "Vc-dimension and shortest path algorithms," in *Automata, Languages and Programming*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 690–699.

[2] ——, "Highway dimension and provably efficient shortest path algorithms," *J. ACM*, vol. 63, no. 5, Dec. 2016.

[3] A. Adamaszek, A. Czumaj, and A. Lingas, "PTAS for k-tour cover problem on the plane for moderately large values of k," in *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5878, Springer, 2009, pp. 994–1003.

[4] S. Arora, "Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems," *J. ACM*, vol. 45, no. 5, pp. 753–782, Sep. 1998.

[5] T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama, "Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, ACM, 1997, pp. 275–283.

[6] A. Becker, "A tight 4/3 approximation for capacitated vehicle routing in trees," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, ser. LIPIcs, vol. 116, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 3:1–3:15.

[7] A. Becker, P. N. Klein, and D. Saulpic, "A quasi-polynomial-time approximation scheme for vehicle routing on planar and bounded-genus graphs," in *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, ser. LIPIcs, vol. 87, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 12:1–12:15.

[8] ——, "Polynomial-time approximation schemes for k-center, k-median, and capacitated vehicle routing in bounded highway dimension," in *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22,*

2018, Helsinki, Finland, ser. LIPIcs, vol. 112, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 8:1–8:15.

[9] A. Becker, P. N. Klein, and A. Schild, "A PTAS for bounded-capacity vehicle routing in planar graphs," in *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11646, Springer, 2019, pp. 99–111.

[10] A. Becker and A. Paul, "A framework for vehicle routing approximation schemes in trees," in *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11646, Springer, 2019, pp. 112–125.

[11] J. Blauth, V. Traub, and J. Vygen, "Improving the approximation ratio for capacitated vehicle routing," *CoRR*, vol. abs/2011.05235, 2020. arXiv: `2011.05235`.

[12] H. L. Bodlaender and T. Hagerup, "Parallel algorithms with optimal speedup for bounded treewidth," in *Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995, Proceedings*, ser. Lecture Notes in Computer Science, vol. 944, Springer, 1995, pp. 268–279.

[13] V. Cohen-Addad, A. Filtser, P. N. Klein, and H. Le, "On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, IEEE, 2020, pp. 589–600.

[14] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, "Treewidth," in *Parameterized Algorithms*. Cham: Springer International Publishing, 2015, pp. 151–244.

[15] M. Cygan, F. Grandoni, S. Leonardi, M. Pilipczuk, and P. Sankowski, "A path-decomposition theorem with applications to pricing and covering on trees," in *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, ser. Lecture Notes in Computer Science, vol. 7501, Springer, 2012, pp. 349–360.

[16] J. H. Dantzig G. B.and Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[17] A. Das and C. Mathieu, "A quasipolynomial time approximation scheme for euclidean capacitated vehicle routing," *Algorithmica*, vol. 73, no. 1, pp. 115–142, 2015.

[18] A. E. Feldmann, W. S. Fung, J. Könemann, and I. Post, "A $(1+\epsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs," in *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 9134, Springer, 2015, pp. 469–480.

[19] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[20] M. Haimovich and A. H. G. R. Kan, "Bounds and heuristics for capacitated routing problems," *Mathematics of Operations Research*, vol. 10, no. 4, pp. 527–542, 1985.

[21] S.-y. Hamaguchi and N. Katoh, "A capacitated vehicle routing problem on a tree," in *Algorithms and Computation*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 399–407.

[22] M. Khachay and R. Dubinin, "PTAS for the euclidean capacitated vehicle routing problem in r^d," in *Discrete Optimization and Operations Research - 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19-23, 2016, Proceedings*, ser. Lecture Notes in Computer Science, vol. 9869, Springer, 2016, pp. 193–205.

[23] M. Khachay and Y. Ogorodnikov, "QPTAS for the CVRP with a moderate number of routes in a metric space of any fixed doubling dimension," in *Learning and Intelligent Optimization - 14th International Conference, LION 14, Athens, Greece, May 24-28, 2020, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 12096, Springer, 2020, pp. 27–32.

[24] M. Khachay, Y. Ogorodnikov, and D. Khachay, "An extension of the das and mathieu QPTAS to the case of polylog capacity constrained CVRP in metric spaces of a fixed doubling dimension," in *Mathematical Optimization Theory and Operations Research - 19th International Conference, MOTOR 2020, Novosibirsk, Russia, July 6-10, 2020, Proceedings*, ser. Lecture Notes in Computer Science, vol. 12095, Springer, 2020, pp. 49–68.

[25] M. Labbé, G. Laporte, and H. Mercure, "Capacitated vehicle routing on trees," *Operations Research*, vol. 39, no. 4, pp. 616–622, 1991.

[26] J. Matoušek, "Embedding finite metric spaces into normed spaces," in *Lectures on Discrete Geometry*. New York, NY: Springer New York, 2002, pp. 355–400.

[27] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*, 2nd. USA: Cambridge University Press, 2017.

[28] K. Talwar, "Bypassing the embedding: Algorithms for low dimensional metrics," in *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '04, Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 281–290.

[29] V. Vazirani, *Approximation Algorithms*. Springer Berlin Heidelberg, 2013.

[30] E. W. Weisstein, "Disk covering problem," *From MathWorld–A Wolfram Web Resource*, 2018.

[31] D. West, *Introduction to Graph Theory*, ser. Featured Titles for Graph Theory. Prentice Hall, 2001.

[32] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2011.