A PTAS for Capacitated Vehicle Routing on Trees*

Claire Mathieu[†] Hang Zhou[‡]

Abstract

We give a polynomial time approximation scheme (PTAS) for the unit demand capacitated vehicle routing problem (CVRP) on trees, for the entire range of the tour capacity. The result extends to the splittable CVRP.

1 Introduction

Given an edge-weighted graph with a vertex called *depot*, a subset of vertices with demands, called *terminals*, and an integer *tour capacity* k, the *capacitated vehicle routing problem (CVRP)* asks for a minimum length collection of tours starting and ending at the depot such that those tours together cover all the demand and the total demand covered by each tour is at most k. In the *unit demand* version, each terminal has unit demand, which is covered by a single tour;[1] in the *splittable* version, each terminal has a positive integer demand and the demand at a terminal may be covered by multiple tours.

The CVRP was introduced by Dantzig and Ramser in 1959 [DR59] and is arguably one of the most important problems in Operations Research. Books have been dedicated to vehicle routing problems, e.g., [TV02, GRW08, CL12, AGM16]. Yet, these problems remain challenging, both from a practical and a theoretical perspective.

Here we focus on the special case when the underlying metric is a tree. That case has been the object of much research. The splittable tree CVRP was proved NP-hard in 1991 [LLM91], so researchers turned to approximation algorithms. Hamaguchi and Katoh [HK98] gave a simple lower bound: every edge must be traversed by enough tours to cover all terminals whose shortest paths to the depot contain that edge. Based on this lower bound, they designed a 1.5-approximation in polynomial time [HK98]. The approximation ratio was improved to $(\sqrt{41}-1)/4$ by Asano, Katoh, and Kawashima [AKK01] and further to 4/3 by Becker [Bec18], both results again based on the lower bound from [HK98]. On the other hand, it was shown in [AKK01] that using this lower bound one cannot achieve an approximation ratio better than 4/3. More recently, researchers tried to go beyond a constant factor so as to get a $(1+\epsilon)$-approximation, at the cost of relaxing some of the constraints. When the tour capacity is allowed to be violated by an ϵ fraction, there is a bicriteria PTAS for the unit demand tree CVRP due to Becker and Paul [BP19]. When the running time is allowed to be quasi-polynomial, Jayaprakash and Salavatipour [JS22] very recently gave a *quasi-polynomial time approximation scheme (QPTAS)* for the unit demand and the splittable versions of the tree CVRP. In this paper, we close this line of research by obtaining a $(1+\epsilon)$-approximation without relaxing any of the constraints – in other words, a *polynomial-time approximation scheme (PTAS)*.

*This is the full version of the extended abstract that was accepted at the 49th EATCS International Colloquium on Automata, Languages, and Programming (ICALP) 2022.

[†]CNRS Paris, France, e-mail: `claire.mathieu@irif.fr`.

[‡]Ecole Polytechnique, Institut Polytechnique de Paris, France, e-mail: `hzhou@lix.polytechnique.fr`.

[1]Thus we may identify the demand coverd with the number of terminals covered.

Theorem 1.1. *There is an approximation scheme for the unit demand capacitated vehicle routing problem (CVRP) on trees with polynomial running time.*

Corollary 1.2. *There is an approximation scheme for the splittable capacitated vehicle routing problem (CVRP) on trees with running time polynomial in the number of vertices n and the tour capacity k.*

To the best of our knowledge, this is the first PTAS for the CVRP in a non-trivial metric and for the entire range of the tour capacity. Previously, PTASs for small capacity as well as QPTASs were given for the CVRP in several metrics, see Section 3.3.

2 Overview of Our Techniques

The main part of our work focuses on the unit demand tree CVRP, and we extend our results to the splittable tree CVRP in the end of this work.

Definition 2.1 (bounded distances)**.** *Let $D_{\min}$ (resp. $D_{\max}$) denote the minimum (resp. maximum) distance between the depot and any terminal in the tree. We say that an instance has* bounded distances *if $D_{\max} < \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min}$.*

Theorem 1.1 follows directly from Theorems 2.2 and 2.3.

Theorem 2.2. *There is a polynomial time $(1 + 4\epsilon)$-approximation algorithm for the unit demand CVRP on the tree T with bounded distances.*

Theorem 2.3. *For any $\rho \geq 1$, if there is a polynomial time ρ-approximation algorithm for the unit demand (resp. splittable, or unsplittable) CVRP on trees with* bounded distances*, then there is a polynomial time $(1 + 5\epsilon)\rho$-approximation algorithm for the unit demand (resp. splittable, or unsplittable) CVRP on trees with* general distances*.*

Theorem 2.3 may be of independent interest for the splittable and the unsplittable versions of the tree CVRP.

2.1 Outline of the PTAS for unit demand instances with bounded distances (Proof of Theorem 2.2)

In Sections 5 to 7, we show that there exists a near-optimal solution with a simple structure, and in Section 8, we use a dynamic program to compute the best solution with that structure.

In Section 5, we consider the *components* of the tree T (defined in Lemma 4.2) and we show that there exists a near-optimal solution such that terminals within each component are visited by a constant $O_\epsilon(1)$ number of tours and that each of those tours visits $\Omega_\epsilon(k)$ terminals in that component (Theorem 5.1). The proof of Theorem 5.1 contains several novelties in our work. We start by defining *large* and *small* subtours inside a component, depending on the number of terminals visited by the subtours (Definition 5.2). To construct a near-optimal solution with that structure, first, we detach small subtours from their initial tours, combine small subtours in the same component, and reallocate the combined subtours to existing tours by a result of Becker and Paul (Lemma 4.6). Then we remove subtours from tours exceeding capacity. To connect the removed subtours to the root of the tree, we include the *spines subtours* (Definition 4.5) of all *internal components* and we obtain a traveling salesman tour (see Fig. 1). Next, we apply the *iterated tour partitioning (ITP)* algorithm on the traveling salesman tour (see Fig. 2), so that we obtain a set of tours each within the tour capacity. Finally, in a postprocessing step, we eliminate the small subtours created due to the ITP algorithm.

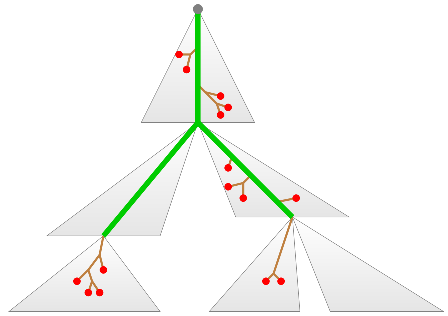

Figure 1: A traveling salesman tour connecting the removed subtours. The brown pieces represent subtours that are removed from the tours exceeding capacity. We add the thick paths (in green) to connect all of those pieces to the root of the tree. The cost of the thick paths is an $O(\epsilon)$ fraction of the optimal cost (Lemma 5.5 and Corollary 5.6), thanks to the bounded distance property. The induced traveling salesman tour spans the removed terminals (in red).

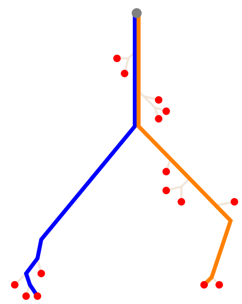

Figure 2: *Iterated tour partitioning (ITP)* on the traveling salesman tour. The two paths (in blue and in orange) represent the connections to the depot added by the ITP algorithm. Their overall cost is an $O(\epsilon)$ fraction of the optimal cost (Lemma 5.7), thanks to the bound on the number of removed terminals (in red) and the bounded distance property.

The complete construction is in Section 5.1; the feasibility of the construction is in Section 5.2; and the analysis on the constructed solution is in Section 5.3, which in particular uses the bounded distance property.

In Section 6, we transform the tree T into a tree $\hat{T}$ that has $O_\epsilon(1)$ levels of components (Fig. 3) and satisfies the following property.

Fact 2.4. *The tree $\hat{T}$ defined in Section 6 can be computed in polynomial time. The components in the tree $\hat{T}$ are the same as those in the tree T. Any solution for the unit demand CVRP on the tree $\hat{T}$ can be transformed in polynomial time into a solution for the unit demand CVRP on the tree T without increasing the cost.*

Thanks to the structure of the near-optimal solution on T (Section 5) and to the bounded distance property (Definition 2.1 and Theorem 2.3), the optimal cost for $\hat{T}$ is increased by an $O(\epsilon)$ fraction compared with the optimal cost for T (Theorem 6.3).

In Section 7, we apply the *adaptive rounding* on the demands of the subtours in a near-optimal solution on $\hat{T}$. We show the existence of a near-optimal solution in which the demand of a subtour can be rounded to the nearest value among a *constant* $O_\epsilon(1)$ number of values. To analyze the adaptive rounding, we observe that an extra cost occurs whenever we detach a subtour and complete it into a separate tour by connecting it to the depot. We bound the extra cost thanks to the structure of a near-

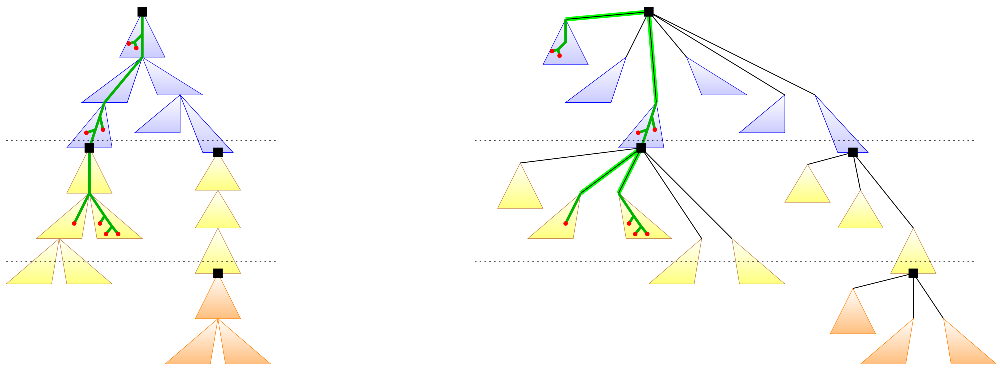

Figure 3: Height reduction for a tree of components. The left figure represents the initial tree of components, where each triangle represents a component. We partition the components into classes (indicated by blue, yellow, and orange), according to the distances from the roots of the components to the root of the tree, and we reduce the height within each class to 1 (right figure), see Section 6. The thick green path in the left figure represents a tour in an optimal solution. The red circular nodes are the terminals visited by that tour. The corresponding tour in the new tree (right figure) spans the same set of terminals.

optimal solution inside components (Section 5), the reduced height of the components (Section 6), as well as the bounded distance property.

In Section 8, we design a *polynomial-time dynamic program* that computes the best solution on the tree $\hat{T}$ that satisfies the constraints on the solution structure imposed by previous sections. The algorithm combines a dynamic program inside components (Section 8.1) and two dynamic programs in subtrees (Section 8.2), see Fig. 4. Thus we obtain the following Theorem 2.5.

Theorem 2.5. *Consider the unit demand CVRP on the tree* $\hat{T}$*. There is a dynamic program that computes in polynomial time a solution with cost at most* $(1+4\epsilon)\cdot\mathrm{opt}$*, where* opt *denotes the optimal cost for the unit demand CVRP on the tree* T*.*

Theorem 2.2 follows directly from Theorem 2.5 and Fact 2.4.

2.2 Reduction from general distances to bounded distances (Proof of Theorem 2.3)

In Section 9, we prove Theorem 2.3. We use Baker's technique to split tours into pieces such that each piece covers terminals that are within a certain range of distances from the depot. This requires duplicating some parts of the tours so that each piece of the tour is connected to the depot.

2.3 Extension to the splittable setting (Proof of Corollary 1.2)

In Section 10, we extend the result in Theorem 1.1 to the splittable setting, thus obtaining Corollary 1.2.

2.4 Open questions

Previously, Jayaprakash and Salavatipour [JS22] extended their QPTAS on trees to QPTASs on graphs of bounded treewidth and beyond, including Euclidean spaces. While some of our techniques extend to those settings, others do not seem to carry over without significant additional ideas, so it is an interesting open question whether the techniques in our paper could be used in the design

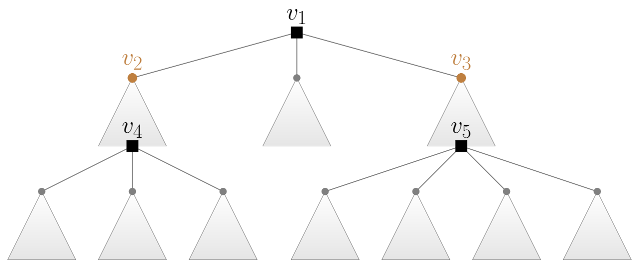

Figure 4: At each vertex of the tree, the dynamic program memorizes the capacities used by the subtours in the subtree and their total cost. Terminals within each component are visited by $O_\epsilon(1)$ tours and each of those tours visits $\Omega_\epsilon(k)$ terminals in that component. Here is an example of the flow of execution in the dynamic program. First, independent computation in each component (Algorithm 2). Next, computation in the subtrees rooted at vertices v_4 and v_5 (Algorithm 4). Then computation for the subtrees rooted at vertices v_2 and v_3 (Algorithm 3). Finally, computation for the subtree rooted at vertex v_1 (Algorithm 4). The output is the best solution in the subtree rooted at v_1. Vertices v_1, v_4, and v_5, whose degrees may be arbitrarily large, are where adaptive rounding of the subtour demands is needed to maintain polynomial time (Algorithm 4). The cost due to the rounding is small thanks to the way components are defined and the bounded distance property, and does not accumulate excessively because the height is bounded (Section 6).

of PTAS algorithms for other metrics, such as graphs of bounded treewidth, planar graphs, and Euclidean spaces.

3 Related Work

Our algorithms build on [JS22] and [BP19] but with the addition of significant new ideas, as we now explain.

3.1 Comparison with the QPTAS in [JS22]

Jayaprakash and Salavatipour noted in [JS22] that

> "*it is not clear if it (the QPTAS) can be turned into a PTAS without significant new ideas.*"

The running time in [JS22] is $n^{O_\epsilon(\log^4 n)}$. Where do those four $\log n$ factors in the exponent come from? At a high level, the QPTAS in [JS22] consists of three parts: (1) reducing the height of the tree; (2) designing a bicriteria QPTAS; (3) going from the bicriteria QPTAS to a true QPTAS. Our approach builds on [JS22] but differs from it in each of the three parts, so that in the end we get rid of all of four $\log n$ factors, hence a PTAS.

(1) Jayaprakash and Salavatipour [JS22] reduce the input tree height from $O(n)$ to $O_\epsilon(\log^2 n)$; whereas instead of the input tree, we consider a tree of components (Lemma 4.2) and reduce its height to $O_\epsilon(1)$, see Fig. 3. Pleasingly, the height reduction (Section 6) is much simpler than in [JS22].

(2) The main technique in [JS22] is to show the existence of a near-optimal solution in which the demand of a subtour can be rounded to the nearest value from a set of only *poly-logarithmic* threshold values. To that end, Jayaprakash and Salavatipour consider the entire range $[1, k]$ of the demands of subtours and partition the subtour demands into *buckets*, resulting in $\Omega_\epsilon(\log k)$ different subtour demands after rounding. In our approach, we define large and small subtours inside components, depending on whether their demands are $\Omega_\epsilon(k)$. Then we transform the solution structure to eliminate small subtours (Section 5), hence only $O_\epsilon(1)$ different subtour demands after rounding. Our analysis

of the adaptive rounding is simpler than in [JS22], and in particular, we do not need the concept of buckets.

(3) Jayaprakash and Salavatipour show that the terminals that are removed from the tours exceeding capacity can probably be covered by duplicating a small random set of tours in the optimal solution. Their approach requires remembering the demands of $\Omega(\log n)$ subtours passing through each edge.[2] To avoid this $\Omega(\log n)$ factor, our approach to cover those terminals is different and uses the ITP algorithm. The analysis of our approach (Sections 5.2 and 5.3) contains several novelties of this paper.

3.2 Comparison with the Bicriteria PTAS in [BP19]

Why is the algorithm in [BP19] a bicriteria PTAS, but not a PTAS?

Becker and Paul [BP19] start by decomposing the tree into *clusters*. (1) They require that each leaf cluster is visited by a single tour. When the violation of the tour capacity is not allowed, this requirement does not preserve a $(1+\epsilon)$-approximate solution, see Fig. 5. (2) They also require that each small internal cluster is visited by a single tour. To that end, they modify the optimal solution by reassigning all terminals of a small cluster to some existing tour at the cost of possibly violating the tour capacity. Such modifications do not seem achievable in the design of a PTAS.

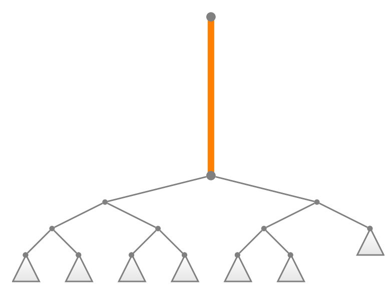

Figure 5: Bad instance for the cluster decomposition in [BP19]. Let $m = 2\lceil \frac{1}{\delta} \rceil + 1$, where $\delta = \Theta(\epsilon)$ is defined in [BP19]. Let the number of terminals in the tree be $2k$. The $2k$ terminals belong to m subtrees of $2k/m$ terminals each (assuming that $2k$ is a multiple of m). Each of the m subtrees (represented by a triangle) is a *leaf cluster* according to the decomposition in [BP19]. The unique edge incident to the root vertex (thick orange edge) has weight 1, and all other edges in the tree have weight 0. The optimal solution consists of two tours (each visiting exactly k vertices), thus has cost 4. When the violation of tour capacity is not allowed and when each leaf cluster is required to be visited by a single tour, the solution contains at least 3 tours, hence a cost of at least 6.

In this paper, we start by defining *components* (Lemma 4.2), inspired by clusters in [BP19]. Unlike [BP19], we allow terminals in any component to be visited by multiple tours. However, allowing many subtours inside a component could result in an exponential running time for a dynamic program. To prevent that, we modify the solution structure inside a component so that the number of subtours becomes bounded (Theorem 5.1). Instead of considering all subtours simultaneously as in [BP19], we distinguish small subtours from large subtours. Inspired by [BP19], we combine small subtours and reallocate them to existing tours such that the violation of the tour capacity is an $O(\epsilon)$ fraction, see Steps 1 to 3 of the construction in Section 5.1. Next, we use the ITP and its postprocessing to reduce the demand of the tours exceeding capacity, which is a novelty in this paper, see Steps 4 to 6 of the construction in Section 5.1. We bound the cost due to the ITP algorithm thanks to the bounded distance property (Theorem 2.3) and to the parameters in our component decomposition

[2]See the proofs of Lemma 2 and Lemma 3 in the full version of [JS22] at `https://arxiv.org/abs/2106.15034`.

that are different from those in [BP19], see Remark 4.3. Besides the above novelties in our approach, the height reduction (Fig. 3, see also Section 6), the adaptive rounding (Section 7), the reduction to bounded distances (Section 9), as well as part of the dynamic program (Section 8) are new compared with [BP19]. These additional techniques are essential in the design of our PTAS, because of the more complicated solution structure inside components in our approach compared with the solution structure inside clusters in [BP19].

3.3 Other Related Work

Constant-factor approximations in general metric spaces The CVRP is a generalization of the *traveling salesman problem (TSP)*. In general metric spaces, Haimovich and Rinnooy Kan [HR85] introduced a simple heuristics, called *iterated tour partitioning (ITP)*. Altinkemer and Gavish [AG90] showed that the approximation ratio of the ITP algorithm for the unit demand and the splittable CVRP is at most $1+\left(1-\frac{1}{k}\right)C_{\text{TSP}}$, where $C_{\text{TSP}} \geq 1$ is the approximation ratio of a TSP algorithm. Bompadre, Dror, and Orlin [BDO06] improved this bound to $1+\left(1-\frac{1}{k}\right)C_{\text{TSP}} - \Omega\left(\frac{1}{k^3}\right)$. The ratio for the unit demand and the splittable CVRP on general metric spaces was recently improved by Blauth, Traub, and Vygen [BTV21] to $1+C_{\text{TSP}}-\epsilon$, for some small constant $\epsilon > 0$.

QPTASs Das and Mathieu [DM15] designed a QPTAS for the CVRP in the Euclidean space; Jayaprakash and Salavatipour [JS22] designed a QPTAS for the CVRP in trees and extended that algorithm to QPTASs in graphs of bounded treewidth, bounded doubling or highway dimension. When the tour capacity is fixed, Becker, Klein, and Saulpic [BKS17] gave a QPTAS for planar graphs and bounded-genus graphs.

PTASs for small capacity In the Euclidean space, there have been PTAS algorithms for the CVRP with small capacity k: work by Haimovich and Rinnooy Kan [HR85], when k is constant; by Asano et al. [AKTT97] extending techniques in [HR85], for $k = O(\log n/\log\log n)$; and by Adamaszek, Czumaj, and Lingas [ACL10], when $k \leq 2^{\log^{f(\epsilon)}(n)}$. For higher dimensional Euclidean metrics, Khachay and Dubinin [KD16] gave a PTAS for fixed dimension ℓ and $k = O(\log^{\frac{1}{\ell}}(n))$. Again when the capacity is bounded, Becker, Klein and Schild [BKS19] gave a PTAS for planar graphs; Becker, Klein, and Saulpic [BKS18] gave a PTAS for graphs of bounded highway dimension; and Cohen-Addad et al. [CFKL20] gave PTASs for bounded genus graphs and bounded treewidth graphs.

Unsplittable CVRP In the *unsplittable* version of the CVRP, every terminal has a positive integer demand, and the entire demand at a terminal should be served by a single tour. On general metric spaces, the best-to-date approximation ratio for the unsplittable CVRP is roughly 3.194 due to the recent work of Friggstad et al. [FMRS21]. For tree metrics, the unsplittable CVRP is APX-hard: indeed, it is NP-hard to approximate the unsplittable tree CVRP to better than a 1.5 factor [GW81] using a reduction from the bin packing problem. Labbé, Laporte and Mercure [LLM91] gave a 2-approximation for the unsplittable tree CVRP. The approximation ratio for the unsplittable tree CVRP was improved to $(1.5+\epsilon)$ very recently by Mathieu and Zhou [MZ22], building upon several techniques in the current paper.

4 Preliminaries

Let T be a rooted tree (V, E) with root $r \in V$ and edge weights $w(u,v) \geq 0$ for all $(u,v) \in E$. The root r represents the *depot* of the tours. Let n denote the number of vertices in V. Let $V' \subseteq V$ denote the set of *terminals*, such that a *token* is placed on each terminal $v \in V'$. Let $k \in [1, n]$ be an integer *capacity* of the tours. The *cost* of a tour t, denoted by $\text{cost}(t)$, is the overall weight of the edges on that tour. We say that a tour *visits* a terminal $v \in V'$ if the tour picks up the token at v.[3]

Definition 4.1 (unit demand tree CVRP). *An instance of the* unit demand *version of the* capacitated vehicle routing problem (CVRP) *on* trees *consists of*

- *an edge weighted* tree $T = (V, E)$ *with* $n = |V|$ *and with* root $r \in V$ *representing the* depot*,*
- *a set* $V' \subseteq V$ *of* terminals,
- *a positive integer* tour capacity k *such that* $k \leq n$*.*

A feasible solution is a set of tours such that

- *each tour starts and ends at* r*,*
- *each tour visits at most* k *terminals,*
- *each terminal is visited by one tour.*

The goal is to find a feasible solution such that the total cost of the tours is minimum.

Let OPT (resp. OPT_1, OPT_2, or OPT_3) denote an optimal (resp. near-optimal) solution to the unit demand CVRP, and let opt (resp. opt_1, opt_2, or opt_3) denote the value of that solution.

Without loss of generality, we assume that every vertex in the input tree T has exactly two children, and that the terminals are the same as the leaf vertices of the tree. Indeed, general instances can be reduced to instances with these properties by inserting edges of weight 0, removing leaf vertices that are not terminals, and slicing out internal vertices of degree two, see, e.g. [BP19] for details.

For any vertex $v \in V$, a *subtour at the vertex* v is a path that starts and ends at v and only visits vertices in the subtree rooted at v. The *demand* of a subtour is the number of terminals visited by that subtour. For each vertex $v \in V$, let $\text{dist}(v)$ denote the distance between v and the depot in the tree T. For technical reasons, we allow *dummy* terminals to be included in the solution at internal vertices of the tree.

Throughout the paper, we define several constants depending on ϵ: Γ (Lemma 4.2), α (Theorem 5.1), H_ϵ (Lemma 6.1), and β (Theorem 7.1). They satisfy the relation that $H_\epsilon \gg \Gamma \gg 1 \gg \epsilon \gg \alpha \gg \beta$.

Decomposition of the Tree into Components

The component decomposition (Lemma 4.2) is inspired by the cluster decomposition by Becker and Paul [BP19]. The proof of Lemma 4.2 is similar to arguments in [BP19]; we give its proof in Appendix A for completeness.

Lemma 4.2. *Let* $\Gamma = \frac{12}{\epsilon}$*. There is a polynomial time algorithm to compute a partition of the edges of the tree* T *into a set* $\mathcal{C}$ *of* components *(see Fig. 6), such that all of the following properties are satisfied:*

- *Every component* $c \in \mathcal{C}$ *is a connected subgraph of* T*; the* root *vertex of the component* c*, denoted by* r_c*, is the vertex in* c *that is closest to the depot.*

[3] Note that a tour might go through a terminal v without picking up the token at v.

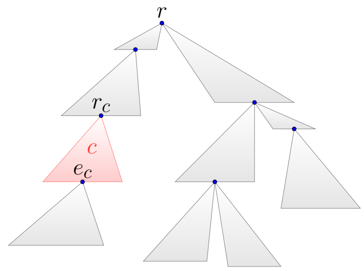

Figure 6: Decomposition into components. In this example, the tree is decomposed into four *leaf* components and six *internal* components. An internal component c has a *root* vertex r_c and an *exit* vertex e_c.

- *We say that a component $c \in \mathcal{C}$ is a* leaf *component if all descendants of r_c in tree T are in c, and is an* internal *component otherwise. A leaf component c interacts with other components at vertex r_c only. An internal component c interacts with other components at two vertices only: at vertex r_c, and at another vertex, called the* exit *vertex of the component c, and denoted by e_c.*
- *Every component $c \in \mathcal{C}$ contains at most $2\Gamma \cdot k$ terminals. We say that a component is* big *if it contains at least $\Gamma \cdot k$ terminals. Each leaf component is big.*
- *If the number of components in $\mathcal{C}$ is strictly greater than one, then we have: (1) there exists a map from all components to big components, such that the image of a component is among its descendants (including itself), and each big component has at most three pre-images; and (2) the number of components in $\mathcal{C}$ is at most $3/\Gamma$ times the total demand in the tree T.*

Remark 4.3. *The root and the exit vertices of components are a rough analog of* portals *used in approximation schemes for other problems: they are places where the dynamic program will gather and synthesize information about partial solutions before passing it on. Compared with [BP19], the decomposition in Lemma 4.2 uses different parameters: the number of terminals inside a* leaf component *is $\Theta(k/\epsilon)$, whereas in [BP19] the number of terminals inside a* leaf cluster *is $\Theta(\epsilon \cdot k)$; the threshold demand to define* big components *is $\Theta(k/\epsilon)$, whereas in [BP19] the threshold demand to define* small clusters *is $\Theta(\epsilon^2 \cdot k)$.*

Definition 4.4 (subtours in components and subtour types)**.** *Let c be any component. A* subtour in the component *c is a path that starts and ends at the root r_c of the component, and such that every vertex on the path is in c. The* type *of a subtour is* "passing" *if c is an internal component and the exit vertex e_c belongs to that subtour; and is* "ending" *otherwise.*

A passing subtour in c is to be combined with a subtour at e_c. In a leaf component, there is no passing subtour.

Definition 4.5 (spine subtour)**.** *For an internal component c, we define the* spine subtour *in the component c, denoted by spine_c, to be the connection (in both directions) between the root vertex r_c and the exit vertex e_c of the component c, without visiting any terminal.*

From the definition, a spine subtour in a component is also a passing subtour in that component.

Without loss of generality, we assume that any subtour in a component c either visits at least one terminal in c or is a spine subtour; that any tour traverses each edge of the tree at most twice (once in each direction); and that any tour contains at most one subtour in any component.[4]

[4] If a tour contains several subtours in a component c, we may combine those subtours into a single subtour in c without increasing the cost of the tour.

Lemma 4.6 (Assignment Lemma, Lemma 1 in [BP19]). *Let $G = (V[G], E[G])$ be an edge-weighted bipartite graph with vertex set $V[G] = A \uplus B$ and edge set $E[G] \subseteq A \times B$, such that each edge $(a,b) \in E[G]$ has a weight $w(a,b) \geq 0$. For each vertex $b \in B$, let $N(b)$ denote the set of vertices $a \in A$ such that $(a,b) \in E[G]$. We assume that $N(b) \neq \emptyset$ and the weight $w(b)$ of the vertex b satisfies $0 \leq w(b) \leq \sum_{a \in N(b)} w(a,b)$. Then there exists a function $f : B \to A$ such that each vertex $b \in B$ is assigned to a vertex $a \in N(b)$ and, for each vertex $a \in A$, we have*

$$\sum_{b \in B | f(b) = a} w(b) - \sum_{b \in B | (a,b) \in E[G]} w(a,b) \leq \max_{b \in B} \{w(b)\}.$$

5 Solutions Inside Components

In this section, we prove Theorem 5.1, which is a main novelty in this paper.

Theorem 5.1. *Let $\alpha = \epsilon^{(\frac{1}{\epsilon}+1)}$. Consider the unit demand CVRP on the tree T with bounded distances. There exist dummy terminals and a solution OPT_1 visiting all of the real and the dummy terminals, such that all of the following holds:*

1. *For each component c, there are at most $\frac{2\Gamma}{\alpha} + 1$ tours visiting terminals in c;*
2. *For each component c and each tour t visiting terminals in c, the number of the terminals in c visited by t is at least $\alpha \cdot k$;*
3. *We have $\mathrm{opt}_1 \leq (1+\epsilon) \cdot \mathrm{opt}$.*

5.1 Construction of OPT_1

Definition 5.2 (large and small subtours). *We say that a subtour is* large *if its demand is at least $\alpha \cdot k$, and* small *otherwise.*

The construction of OPT_1, starting from an optimal solution OPT, is in several steps.

Step 1: Detaching small subtours

Prune each tour of OPT so that it only visits the terminals that do *not* belong to a small subtour in any component, and is minimal. In other words, if a tour t in OPT contains a small ending subtour t_e in a component c, then we remove t_e from t; and if a tour t in OPT contains a non-spine small passing subtour t_p in a component c, then we remove t_p from t, except for the spine subtour of c.

Let A denote the set of the resulting tours. Note that each tour in A is connected. The removed pieces of a non-spine small passing subtour t_p may be disconnected from one another.

The parts of OPT that have been pruned consist of the set $\mathcal{E}$, each element being a small ending subtours in a component, and of the set $\mathcal{P}$, each element being a group of pieces in a component obtained from a non-spine small passing subtour by removing the corresponding spine subtour. The *demand* of a group of pieces in $\mathcal{P}$ is the number of terminals in all of the pieces in that group.

Step 2: Combining small subtours within components

- For each component c, repeatedly concatenate subtours in c from the set $\mathcal{E}$ so that in the end, all resulting subtours in c from the set $\mathcal{E}$ have demands between $\alpha \cdot k$ and $2\alpha \cdot k$ except for at most one subtour.

- For each component c, repeatedly merge groups in c from the set $\mathcal{P}$ so that in the end, all resulting groups in c from the set $\mathcal{P}$ have demands between $\alpha \cdot k$ and $2\alpha \cdot k$ except for at most one group.

Let $\mathcal{E}'$ and $\mathcal{P}'$ denote the corresponding sets after the modifications for all components c. Let $B = \mathcal{E}' \cup \mathcal{P}'$. An *element* of B is either a *subtour* or a *group of pieces* in a component c. For each component c, all elements of B in the component c have demands between $\alpha \cdot k$ and $2\alpha \cdot k$ except for at most two elements with smaller demands.

Step 3: Reassigning small subtours

Construct a bipartite graph with vertex sets A and B and with edge set E. Let a be any tour in A, and let a_0 denote the corresponding tour in OPT. Let b be any element in B. The set E contains an edge between a and b if and only if the element b contains terminals from a_0; the weight of the edge (a, b) is the number of terminals in both b and a_0. By Lemma 4.6, there exists an assignment $f : B \to A$ such that each element $b \in B$ is assigned to a tour $a \in A$ with $(a, b) \in E$ and that, for each tour $a \in A$, the demand of a plus the overall demand of the elements $b \in B$ that are assigned to a is at most the demand of the corresponding tour a_0 plus the maximum demand of any element in B.

Let A_1 denote the set of *pseudo-tours* induced by the assignment f. Each pseudo-tour in A_1 is the union of a tour $a \in A$ and the elements $b \in B$ that are assigned to a.

Step 4: Correcting tour capacities

For each pseudo-tour a_1 in A_1, if the demand of a_1 exceeds k, we repeatedly remove an element $b \in B$ from a_1, until the demand of a_1 is at most k.

Let A_2 denote the resulting set of pseudo-tours. Every pseudo-tour in A_2 is a connected tour of demand at most k (Lemma 5.3). Let $B^* \subseteq B$ denote the set of the removed elements $b \in B$. The elements in B^* are represented by the small pieces in Fig. 1 (Page 3).

Step 5: Creating additional tours

We connect the elements of B^* to the depot by creating additional tours as follows.

- Let Q denote the union of the spine subtours for all internal components. Q is represented by the green thick paths in Fig. 1. Let X denote a multi-subgraph of T that is the union of the elements in B^* and the edges in Q. Observe that each element in B^* is connected to the depot through edges in Q. Thus X is connected, and induces a traveling salesman tour t_{TSP} visiting all terminals in B^*. Without loss of generality, we assume that, for any component c, if t_{TSP} visits terminals in c, then those terminals belong to a single subpath p_c of t_{TSP}, such that p_c does not visit any terminal from other components.

- If the traveling salesman tour t_{TSP} is within the tour capacity, then let A_3 denote the set consisting of a single tour t_{TSP}. Otherwise, we apply the *iterated tour partitioning (ITP)* algorithm [HR85] on t_{TSP}: we partition t_{TSP} into segments with exactly k terminals each, except possibly the last segments containing less than k terminals. For each segment, we connect its endpoints to the depot so as to make a tour, see Fig. 2 (Page 3). Let A_3 denote the resulting set of tours.

Let $A_4 = A_2 \cup A_3$.

Step 6: Postprocessing

For each component c, we rearrange the small subtours in A_4 as follows.

- For each tour t in A_4 that contains a small subtour in c, letting t_c denote this small subtour, if t_c is an ending subtour, we remove t_c from t; if t_c is a passing subtour, we remove t_c from t, except for the spine subtour in c. The total demand of all of the removed small subtours in c is at most k (Lemma 5.4).
- We create an additional tour t_c^* from the depot that connects all of the removed small subtours in c. If the demand of t_c^* is less than $\alpha \cdot k$, then we include dummy terminals at r_c into the tour t_c^* so that its demand is exactly k.

Let A_5 denote the resulting set of tours after removing small subtours from A_4. Let A_6 denote the set of newly created tours $\{t_c^*\}_{c \in \mathcal{C}}$.

Finally, let $\mathrm{OPT}_1 = A_5 \cup A_6$.

In Section 5.2, we show that OPT_1 is a feasible solution to the unit demand tree CVRP; in Section 5.3, we prove the three properties of OPT_1 in the claim of Theorem 5.1.

5.2 Feasibility of the Construction

Lemma 5.3. *Every pseudo-tour in A_2 is a connected tour of demand at most k.*

Proof. Let a_2 be any pseudo-tour in A_2. By the construction in Step 4, the demand of a_2 is at most k. It suffices to show that a_2 is a connected tour.

Observe that a_2 is the union of a tour $a \in A$ and some elements $b_1, \dots, b_q$ from B, for $q \geq 0$. From the construction, any tour $a \in A$ is connected. Consider any b_i for $i \in [1, q]$. Observe that $f(b_i) = a$, so the edge (a, b_i) belongs to the bipartite graph (A, B). If b_i is a subtour at r_c for some component c, then r_c must belong to the tour a; and if b_i is a group of pieces in some component c, then the spine subtour of c must belong to the tour a. So the union of a and b_i is connected. Thus $a_2 = a \cup b_1 \cup \cdots \cup b_q$ is a connected tour. □

Lemma 5.4. *In any component c, the total demand of all of the removed small subtours in c at the beginning of Step 6 is at most k.*

Proof. Let c be any component. The key is to show that the number of non-spine small subtours in c that are contained in tours in A_4 is at most 4. Since $A_4 = A_2 \cup A_3$, we analyze the number of non-spine small subtours in c that are contained in tours in A_2 and in A_3, respectively.

The tours in A_2 contain at most two non-spine small subtours in c, since at most two elements of B in component c have demands less than $\alpha \cdot k$.

We claim that the tours in A_3 contain at most two non-spine small subtours in c. If A_3 consists of a single tour t_{TSP}, the claim follows trivially since any tour contains at most one subtour in c from our assumption. Next, we consider the case when A_3 is generated by the ITP algorithm. From our assumption, if t_{TSP} visits terminals in c, then those terminals belong to a single subpath p_c of t_{TSP}, such that p_c does not visit any terminal from other components. By applying the ITP algorithm on t_{TSP}, we obtain a collection of segments. All segments that intersect p_c visit exactly k terminals in c, except for possibly the first and the last of those segments visiting less than k terminals in c. Hence at most two non-spine small subtours in c among the tours in A_3.

Therefore, the number of non-spine small subtours in c in the solution A_4 is at most 4. Since each small subtour has demand at most $\alpha \cdot k$, the total demand of the removed small subtours is at most $4 \cdot \alpha \cdot k < k$. □

5.3 Analysis of OPT_1

Let $c \in \mathcal{C}$ be any component. From Lemma 4.2, c contains at most $2\Gamma \cdot k$ real terminals. Each tour in A_5 visiting terminals in c visits at least $\alpha \cdot k$ real terminals in c, so there are at most $\frac{2\Gamma}{\alpha}$ tours in A_5 visiting terminals in c. There is a single tour in A_6, the tour t_c^*, that visits terminals in c. Hence the first property of the claim. From the construction of t_c^*, the second property of the claim follows.

It remains to analyze the cost of OPT_1. Compared with OPT, the extra cost in OPT_1 is due to Step 5 and Step 6 of the construction. This extra cost consists of the cost of the edges in Q (Step 5), the cost in the ITP algorithm to connect the endpoints of all segments to the depot (Step 5), and the cost of the postprocessing (Step 6), which we bound in Corollary 5.6 and Lemmas 5.7 and 5.8, respectively. Both Corollary 5.6 and Lemma 5.8 are based on Lemma 5.5.

Lemma 5.5. *We have* $\displaystyle\sum_{\text{component } c} \mathrm{dist}(r_c) \le \frac{\epsilon}{8} \cdot \mathrm{opt}.$

Proof. For any edge e in T, let u and v denote the two endpoints of e such that u is the parent of v. Let T_e denote the subtree of T rooted at v. Let n_e denote the number of terminals in T_e. From the lower bound in [HK98], we have

$$\mathrm{opt} \ge \sum_{e \in T} 2 \cdot w(e) \cdot \frac{n_e}{k}.$$

Since each big component contains at least $\Gamma \cdot k$ terminals, we have

$$n_e \ge \sum_{\text{big component } c \subseteq T_e} \Gamma \cdot k.$$

Thus

$$\begin{aligned}\mathrm{opt} &\ge \sum_{e \in T} 2 \cdot w(e) \cdot \sum_{\text{big component } c \subseteq T_e} \Gamma \\ &= \sum_{\text{big component } c} \Gamma \cdot \sum_{e \in T \text{ such that } c \subseteq T_e} 2 \cdot w(e) \\ &= \sum_{\text{big component } c} \Gamma \cdot 2 \cdot \mathrm{dist}(r_c).\end{aligned}$$

From Lemma 4.2, there exists a map from all components to big components such that the image of a component is among its descendants (including itself) and each big component has at most three pre-images. Thus

$$3 \cdot \sum_{\text{big component } c} \mathrm{dist}(r_c) \ge \sum_{\text{component } c} \mathrm{dist}(r_c).$$

Therefore,

$$\mathrm{opt} \ge \frac{2 \cdot \Gamma}{3} \cdot \sum_{\text{component } c} \mathrm{dist}(r_c).$$

The claim follows since $\Gamma = \frac{12}{\epsilon}$. □

Corollary 5.6. *We have* $\mathrm{cost}(Q) \le \frac{\epsilon}{4} \cdot \mathrm{opt}$.

Proof. Observe that every edge in Q belongs to the connection (in both directions) between the depot and the root of some component c. By Lemma 5.5, we have

$$\mathrm{cost}(Q) \le 2 \cdot \sum_{\text{component } c} \mathrm{dist}(r_c) \le \frac{\epsilon}{4} \cdot \mathrm{opt}.$$

□

Lemma 5.7. *Let Δ_1 denote the cost in the ITP algorithm to connect the endpoints of all segments to the depot in Step 5. We have $\Delta_1 \le \frac{\epsilon}{4} \cdot \mathrm{opt}$.*

Proof. Let n' denote the number of terminals in the tree T. First, we show that the number of terminals on t_{TSP} is at most $4\alpha \cdot n'$. Observe that the number of terminals on t_{TSP} is the overall removed demand in Step 4. Consider any pseudo-tour $a_1 \in A_1$ whose demand exceeds k. Let a_0 denote the corresponding tour in OPT. By the construction, the demand of a_1 is at most the demand of a_0 plus the maximum demand of any element in B. Since the demand of a_0 is at most k and the demand of any element in B is at most $2\alpha \cdot k$, the demand of a_1 is at most $k + 2\alpha \cdot k$. Let a_2 denote the corresponding tour after the correction of capacity in Step 4. Since any element in B has demand at most $2\alpha \cdot k$, the demand of a_2 is at least $k - 2\alpha \cdot k$. So the total removed demand from a_1 in Step 4 is at most $4\alpha \cdot k$. There are at most $\frac{n'}{k}$ pseudo-tours $a_1 \in A_1$ whose demands exceed k. Summing over all those pseudo-tours, the overall removed demand in Step 4 is at most $\frac{n'}{k} \cdot 4\alpha \cdot k = 4\alpha \cdot n'$. Hence the number of terminals on t_{TSP} is at most $4\alpha \cdot n'$.

If $4\alpha \cdot n' \le k$, then t_{TSP} is within the tour capacity, so the ITP algorithm is not applied and $\Delta_1 = 0$. It remains to consider the case in which $4\alpha \cdot n' > k$. By the construction in the ITP algorithm, every segment visits exactly k terminals except possibly the last segment. Thus the number ℓ_{ITP} of resulting tours in the ITP algorithm is

$$\ell_{\mathrm{IPT}} \le \frac{4\alpha \cdot n'}{k} + 1. \tag{1}$$

Since $4\alpha \cdot n' > k$, we have $\ell_{\mathrm{IPT}} < \frac{8\alpha \cdot n'}{k}$. Since $\Delta_1 \le \ell_{\mathrm{ITP}} \cdot 2 \cdot D_{\max}$ and using Definition 2.1 and the definition of α in the claim of Theorem 5.1, we have

$$\Delta_1 < \frac{8\alpha \cdot n'}{k} \cdot 2 \cdot \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min} < \frac{\epsilon}{2} \cdot \frac{n'}{k} \cdot D_{\min}.$$

On the other hand, the solution OPT consists of at least $\frac{n'}{k}$ tours, so $\mathrm{opt} \ge \frac{2n'}{k} \cdot D_{\min}$. Therefore, $\Delta_1 \le \frac{\epsilon}{4} \cdot \mathrm{opt}$. □

Lemma 5.8. *Let Δ_2 denote the cost of the postprocessing (Step 6). Then $\Delta_2 \le \frac{\epsilon}{2} \cdot \mathrm{opt}$.*

Proof. For each leaf component c, the cost to connect the small subtours in c to the depot in Step 6 is at most $2 \cdot \mathrm{dist}(r_c)$; and for each internal component c, the cost to connect the small subtours in c to the depot is at most $2 \cdot \mathrm{dist}(r_c) + \mathrm{cost}(\mathrm{spine}_c)$. Summing over all components, we have

$$\Delta_2 \le \sum_{\text{component } c} 2 \cdot \mathrm{dist}(r_c) + \sum_{\text{internal component } c} \mathrm{cost}(\mathrm{spine}_c).$$

By Lemma 5.5, we have

$$\sum_{\text{component } c} 2 \cdot \mathrm{dist}(r_c) \le \frac{\epsilon}{4} \cdot \mathrm{opt}$$

and

$$\sum_{\text{internal component } c} \mathrm{cost}(\mathrm{spine}_c) = \mathrm{cost}(Q) \le \frac{\epsilon}{4} \cdot \mathrm{opt}.$$

Thus $\Delta_2 \le \frac{\epsilon}{2} \cdot \mathrm{opt}$. □

From Corollary 5.6 and Lemmas 5.7 and 5.8, we have $\mathrm{opt}_1 \le \mathrm{opt} + \mathrm{cost}(Q) + \Delta_1 + \Delta_2 \le (1+\epsilon) \cdot \mathrm{opt}$. This completes the proof for Theorem 5.1.

6 Height Reduction

In this section, we transform the tree T into a tree $\hat{T}$ so that $\hat{T}$ has $O_\epsilon(1)$ levels of components, see Fig. 3. We assume that the tree T has bounded distances. To begin with, we partition the components according to the distances from their roots to the depot.

Lemma 6.1. *Let $\tilde{D} = \alpha \cdot \epsilon \cdot D_{\min}$. Let $H_\epsilon = (\frac{1}{\epsilon})^{\frac{2}{\epsilon}+1}$. For each $i \in [1, H_\epsilon]$, let $\mathcal{C}_i \subseteq \mathcal{C}$ denote the set of components $c \in \mathcal{C}$ such that $\text{dist}(r_c) \in \left[(i-1) \cdot \tilde{D}, i \cdot \tilde{D}\right)$. Then any component $c \in \mathcal{C}$ belongs to a set $\mathcal{C}_i$ for some $i \in [1, H_\epsilon]$.*

Proof. Let $c \in \mathcal{C}$ be any component. We have

$$\text{dist}(r_c) \leq D_{\max} < \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min} = H_\epsilon \cdot \tilde{D},$$

where the second inequality follows from Definition 2.1, and the equality follows from the definition of α in Theorem 5.1 and the definitions of $\tilde{D}$ and H_ϵ. Thus there exists $i \in [1, H_\epsilon]$ such that $c \in \mathcal{C}_i$. $\square$

Definition 6.2 (maximally connected sets and critical vertices)**.** *We say that a set of components $\tilde{\mathcal{C}} \subseteq \mathcal{C}_i$ is* maximally connected *if the components in $\tilde{\mathcal{C}}$ are connected to each other and $\tilde{\mathcal{C}}$ is maximal within $\mathcal{C}_i$. For a maximally connected set of components $\tilde{\mathcal{C}} \subseteq \mathcal{C}_i$, we define the* critical vertex *of $\tilde{\mathcal{C}}$ to be the root vertex of the component $c \in \tilde{\mathcal{C}}$ that is closest to the depot.*

Fig. 3 (Page 4) is an example with three levels of components: $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, indicated by different colors. There are four maximally connected sets of components. The critical vertices are represented by rectangular nodes.

Algorithm 1 Construction of the tree $\hat{T}$ (see Fig. 3).

```
for each i ∈ [1, H_ε] do
    for each maximally connected set of components C̃ ⊆ C_i do
        z ← critical vertex of C̃
        for each component c ∈ C̃ do
            δ ← r_c-to-z distance in T
            Split the tree T at the root vertex r_c of the component c      ▷ vertex r_c is duplicated
            Add an edge between the root of the component c and z with weight δ
T̂ ←the resulting tree
```

Let $\hat{T}$ be the tree constructed in Algorithm 1. We observe that Algorithm 1 is in polynomial time, and Fact 2.4 follows from the construction. We show in Theorem 6.3 that the optimal cost for $\hat{T}$ is increased by an $O(\epsilon)$ fraction compared with the optimal cost for T.

Theorem 6.3. *Consider the unit demand CVRP on the tree $\hat{T}$. There exist dummy terminals and a solution OPT_2 visiting all of the real and the dummy terminals, such that all of the following holds:*

1. *For each component c, there are at most $\frac{2\Gamma}{\alpha} + 1$ tours visiting terminals in c;*
2. *For each component c and each tour t visiting terminals in c, the number of the terminals in c visited by t is at least $\alpha \cdot k$;*
3. *We have $\text{opt}_2 < (1 + 3\epsilon) \cdot \text{opt}$, where opt denotes the optimal cost for the unit demand CVRP on the tree T.*

In the rest of the section, we prove Theorem 6.3.

6.1 Construction of OPT_2

Consider any tour t in OPT_1. Let U denote the set of terminals visited by t.[5] We define the tour $\hat{t}$ as the minimal tour in the tree $\hat{T}$ that spans all terminals in U, see Fig. 3. Let OPT_2 denote the set of the resulting tours on the tree $\hat{T}$ constructed from every tour t in OPT_1. Then OPT_2 is a feasible solution to the unit demand CVRP on $\hat{T}$.

6.2 Analysis of OPT_2

The first two properties in Theorem 6.3 follow from the construction and Theorem 5.1.

In the rest of the section, we analyze the cost of OPT_2.

Lemma 6.4. *Let t denote any tour in OPT_1. Let $\hat{t}$ denote the corresponding tour in OPT_2. Then $\mathrm{cost}(\hat{t}) \leq (1+\epsilon)\cdot \mathrm{cost}(t)$.*

Proof. We follow the notation on U from Section 6.1. Let $\mathcal{C}(U)$ denote the set of components $c \in \mathcal{C}$ that contains a (possibly spine) subtour of $\hat{t}$. Observe that the cost of $\hat{t}$ consists of the following two parts:

1. for each component $c \in \mathcal{C}(U)$, the cost of the subtour in c from the tour t; we *charge* that cost to the subtour in t;
2. for each component $c \in \mathcal{C}(U)$, the cost of the edge (r_c, z), where z denotes the father vertex of r_c in $\hat{T}$. Note that z is a critical vertex on $\hat{t}$. We analyze that cost over all components $c \in \mathcal{C}(U)$ as follows.

Let $Z \subseteq V$ denote the set of critical vertices $z \in V$ on $\hat{t}$. For any critical vertex $z \in Z$, let $Y(z)$ denote the set of edges (z, v) in the tree $\hat{T}$ such that v is a child of z and that the edge (z, v) belongs to the tour $\hat{t}$. The overall cost of the second part is the total cost of the edges in $Y(z)$ for all $z \in Z$.

Fix a critical vertex $z \in Z$. Let (z, v_1) denote the edge in $Y(z)$ such that $\mathrm{dist}(v_1)$ is minimized, breaking ties arbitrarily. From the minimality of $\mathrm{dist}(v_1)$, the z-to-v_1 path in T does not go through any component in $\mathcal{C}(U)$. From the construction, the cost of the edge (z, v_1) in $\hat{T}$ equals the cost of the z-to-v_1 path in T. It is easy to see that the z-to-v_1 path in T belongs to the tour t. Indeed, tour $\hat{t}$ traverses the edge (z, v_1) on its way to visit some terminals of U in the subtree rooted at v_1. In order to visit the corresponding terminals in T, tour t must traverse the z-to-v_1 path. We *charge* the cost of the edge (z, v_1) in $\hat{T}$ to the z-to-v_1 path in T. Next, we analyze the cost due to the other edges in $Y(z)$. Consider one such edge (z, v). From the construction, there exists $i \in [1, H_\epsilon]$, such that both $\mathrm{dist}(z)$ and $\mathrm{dist}(v)$ belong to $\left[(i-1)\cdot \tilde{D}, i\cdot \tilde{D}\right)$. Thus the cost of the z-to-v path in T equals $\mathrm{dist}(v) - \mathrm{dist}(z) < \tilde{D}$, so the extra cost in $\hat{t}$ due to the edge (z, v) is at most $2\cdot\tilde{D}$ (for both directions). Therefore, the extra cost in $\hat{t}$ due to those $|Y(z)|-1$ edges in $Y(z)$ is at most $2\cdot\tilde{D}\cdot(|Y(z)|-1)$.

Summing over all vertices $z \in Z$, and observing that all charges are to disjoint parts of t, we have

$$\mathrm{cost}(\hat{t}) \leq \mathrm{cost}(t) + 2\cdot\tilde{D}\cdot\sum_{z\in Z}(|Y(z)|-1). \tag{2}$$

It remains to bound $\sum_{z\in Z}(|Y(z)|-1)$. The analysis uses the following basic fact in trees.

Fact 6.5. *Let H be a tree with L leaves. For each vertex u in H, let $m(u)$ denote the number of children of u in H. Then*

$$\sum_{u\in H}(m(u)-1) \leq L-1.$$

[5] We assume that t is a minimal tour in T spanning all terminals in U.

We construct a tree H as follows. Starting from the tree spanning U in $\hat{T}$, we contract vertices in each component $c \in \mathcal{C}(U)$ into a single vertex; let H denote the resulting tree. It is easy to see that each leaf in H corresponds to a component $c \in \mathcal{C}(U)$ that contains at least one terminal in U (using the definition of $\mathcal{C}(U)$ and the fact that any descending component of c do not belong to $\mathcal{C}(U)$). From the second property of Theorem 6.3 (which follows from Theorem 5.1), terminals in U belong to at most $1/\alpha$ components. Thus, by Fact 6.5 we have

$$\sum_{z \in Z} (|Y(z)| - 1) \leq (1/\alpha) - 1.$$

Combined with Eq. (2), we have

$$\text{cost}(\hat{t}) - \text{cost}(t) < 2 \cdot \tilde{D} \cdot (1/\alpha) = 2 \cdot \alpha \cdot \epsilon \cdot D_{\min} \cdot (1/\alpha) = 2\epsilon \cdot D_{\min},$$

using the definition of $\tilde{D}$ in Lemma 6.1. Since $\text{cost}(t) \geq 2 \cdot D_{\min}$, the claim follows. □

Applying Lemma 6.4 on each tour t in OPT_1 and summing, we have $\text{opt}_2 \leq (1+\epsilon) \cdot \text{opt}_1$. By Theorem 5.1, $\text{opt}_1 \leq (1+\epsilon) \cdot \text{opt}$, thus $\text{opt}_2 \leq (1+3\epsilon) \cdot \text{opt}$. This completes the proof of Theorem 6.3.

7 Adaptive Rounding on the Subtour Demands

In this section, we prove Theorem 7.1. We use the adaptive rounding to show that, in a near-optimal solution, the demands of the subtours at any critical vertex are from a set of $O_\epsilon(1)$ values. This property enables us to later guess those values in polynomial time by a dynamic program (see Section 8).

Theorem 7.1. *Let $\beta = \frac{1}{4} \cdot \epsilon^{(\frac{4}{\epsilon}+1)}$. Consider the unit demand CVRP on the tree $\hat{T}$. There exist dummy terminals and a solution OPT_3 visiting all of the real and the dummy terminals, such that all of the following holds:*

1. *For each component c, there are at most $\frac{2\Gamma}{\alpha} + 1$ tours visiting terminals in c;*
2. *For each critical vertex z, there exist $\frac{1}{\beta}$ integer values in $[\alpha \cdot k, k]$ such that the demands of the subtours at the children of z are among these values;*
3. *We have $\text{opt}_3 < (1+4\epsilon) \cdot \text{opt}$, where opt denotes the optimal cost for the unit demand CVRP on the tree T.*

7.1 Construction of OPT_3

We construct the solution OPT_3 by modifying the solution OPT_2.

Let $I \subseteq V$ denote the set of vertices $v \in V$ that is either the root of a component or a critical vertex. Consider any vertex $v \in I$ in the bottom up order. Let $\text{OPT}_2(v)$ denote the set of subtours at v in OPT_2. We construct a set $A(v)$ of subtours at v satisfying the following invariants:

- the subtours in $A(v)$ have a one-to-one correspondence with the subtours in $\text{OPT}_2(v)$; and
- the demand of each subtour of $A(v)$ is at most that of the corresponding subtour in $\text{OPT}_2(v)$.

The construction of $A(v)$ is according to one of the following three cases on v.

Case 1: v is the root vertex r_c of a leaf component c in $\hat{T}$

Let $A(v) = \text{OPT}_2(v)$.

Case 2: v is the root vertex r_c of an internal component c in $\hat{T}$

For each subtour $a \in \mathrm{OPT}_2(v)$, if a contains a subtour at the exit vertex e_c of component c, letting t denote this subtour and t' denote the subtour in $A(e_c)$ corresponding to t, we replace the subtour t in a by the subtour t'. Let $A(v)$ be the resulting set of subtours at v.

Case 3: v is a critical vertex in $\hat{T}$

We apply the technique of the *adaptive rounding*, previously used by Jayaprakash and Salavatipour [JS22] in their design of a QPTAS the tree CVRP. The idea is to *round up* the demands of the subtours at the children of v so that the resulting demands are among $\frac{1}{\beta}$ values.

Let $r_1, \ldots, r_m$ be the children of v in $\hat{T}$. For each subtour $a \in \mathrm{OPT}_2(v)$ and for each $i \in [1, m]$, if a contains a subtour at r_i, letting t denote this subtour and t' denote the subtour in $A(r_i)$ corresponding to t, we replace t in a by t'. Let $A_1(v)$ denote the resulting set of subtours at v.

Let W_v denote the set of the subtours at the children of v in $A_1(v)$, i.e., $W_v = A(r_1) \cup \cdots \cup A(r_m)$. If $|W_v| \leq \frac{1}{\beta}$, let $A(v) = A_1(v)$. In the following, we consider the non-trivial case when $|W_v| > \frac{1}{\beta}$. We sort the subtours in W_v in non-decreasing order of their demands, and partition these subtours into $\frac{1}{\beta}$ groups of equal cardinality.[6] We *round* the demands of the subtours in each group to the maximum demand in that group. The demand of a subtour is increased to the rounded value by adding *dummy* terminals at the children of v. We rearrange the subtours in W_v as follows.

- Each subtour $t \in W_v$ in the last group is discarded, i.e., detached from the subtour in $A_1(v)$ to which it belongs.
- Each subtour $t \in W_v$ in other groups is associated in a one-to-one manner to a subtour $t' \in W_v$ in the next group. Letting a (resp. a') denote the subtour in $A_1(v)$ to which t (resp. t') belongs, we detach t from a and reattach t to a'.

Let $A(v)$ be the set of the resulting subtours at v after the rearrangement for all $t \in W_v$.

For each subtour t that is discarded in the construction, we complete t into a separate tour by adding the connection (in both directions) to the depot. Let B denote the set of these newly created tours. Let $\mathrm{OPT}_3 = A(r) \cup B$.

It is easy to see that OPT_3 is a feasible solution to the unit demand CVRP, i.e., each tour in OPT_3 is connected and visits at most k terminals, and each terminal is covered by some tour in OPT_3.

7.2 Analysis of OPT_3

From the construction, in any component $c \in \mathcal{C}$, the non-spine subtours in OPT_3 are the same as those in OPT_2. From Theorem 6.3, we obtain the first property in Theorem 7.1, and in addition, each subtour at a child of a critical vertex in OPT_2 has demand at least $\alpha \cdot k$. The second property of the claim follows from the construction of OPT_3.

It remains to analyze the cost of OPT_3. Let $\Delta = \mathrm{opt}_3 - \mathrm{opt}_2$. Observe that Δ is due to adding connections to the depot to create the tours in the set B.

Fix any $i \in [1, H_\epsilon]$. Let $Z \subseteq V$ denote the set of vertices $v \in V$ such that v is the critical vertex of a maximally connected component $\tilde{\mathcal{C}} \subseteq \mathcal{C}_i$. For any $v \in Z$, we analyze the number of discarded subtours in the set W_v defined in Section 7.1. If $|W_v| \leq \frac{1}{\beta}$, there is no discarded subtour in W_v; if $|W_v| > \frac{1}{\beta}$, the number of discarded subtours in W_v is $\lceil \beta \cdot |W_v| \rceil < \beta \cdot |W_v| + 1 < 2\beta \cdot |W_v|$. Let W

[6] We add empty subtours to the first groups if needed in order to achieve equal cardinality among all groups.

denote the disjoint union of W_v for all vertices $v \in Z$. Thus W contains at most $2\beta \cdot |W|$ discarded subtours. Let Δ_i denote the cost to connect the discarded subtours in W to the depot. We have

$$\Delta_i \le 2\beta \cdot |W| \cdot 2 \cdot D_{\max} < \frac{1}{2} \cdot \epsilon^{(\frac{4}{\epsilon}+1)} \cdot |W| \cdot 2 \cdot \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min} = \frac{\epsilon}{H_\epsilon} \cdot \alpha \cdot |W| \cdot D_{\min}, \tag{3}$$

where the second inequality follows from the definition of β (Theorem 7.1) and Definition 2.1, and the equality follows from the definitions of α (Theorem 5.1) and of H_ϵ (Lemma 6.1). From the second property of the claim, each subtour in W has demand at least $\alpha \cdot k$, so there are at least $\alpha \cdot |W|$ tours in OPT_3. Any tour in OPT_3 has cost at least $2 \cdot D_{\min}$, so we have

$$\mathrm{opt}_3 \ge 2 \cdot \alpha \cdot |W| \cdot D_{\min}. \tag{4}$$

From Eqs. (3) and (4), we have

$$\Delta_i < \frac{\epsilon}{2 \cdot H_\epsilon} \cdot \mathrm{opt}_3.$$

Summing over all integers $i \in [1, H_\epsilon]$, we have $\Delta = \sum_i \Delta_i \le \frac{\epsilon}{2} \cdot \mathrm{opt}_3$. Thus

$$\mathrm{opt}_3 \le \frac{2}{2-\epsilon} \cdot \mathrm{opt}_2.$$

By Theorem 6.3, $\mathrm{opt}_2 \le (1+3\epsilon) \cdot \mathrm{opt}$. Therefore, $\mathrm{opt}_3 \le (1+4\epsilon) \cdot \mathrm{opt}$. This completes the proof of Theorem 7.1.

8 Dynamic Programming

In this section, we show Theorem 2.5. In our dynamic program, we consider all feasible solutions on the tree $\hat{T}$ satisfying the properties of OPT_3 in Theorem 7.1, and we output the solution with minimum cost. The first property of Theorem 7.1 is used in Section 8.1 in the computation of solutions inside components, and the second property of Theorem 7.1 is used in Section 8.2 in the computation of solutions in the subtrees rooted at critical vertices. These two properties ensure the polynomial running time of the dynamic program. The cost of the output solution is at most the cost of OPT_3, which is at most $(1+4\epsilon)$ times the optimal cost on the tree T by the third property of Theorem 7.1.

8.1 Local Configurations

In this subsection, we compute values at *local configurations* (Definition 8.1), which are solutions restricted locally to a component. We require that the terminals in any component are visited by at most $\frac{2\Gamma}{\alpha}+1$ tours, using the first property in Theorem 7.1. Thus the number of local configurations is polynomially bounded.

Definition 8.1 (local configurations)**.** *Let $c \in \mathcal{C}$ be any component. A* local configuration (v, A) *is defined by a vertex $v \in c$ and a list A of $\ell(A)$ pairs $(s_1, b_1), (s_2, b_2), \ldots, (s_{\ell(A)}, b_{\ell(A)})$ such that*

- $\ell(A) \le \frac{2\Gamma}{\alpha} + 1$;
- *for each $i \in [1, \ell(A)]$, s_i is an integer in $[0, k]$ and $b_i \in \{$"passing", "ending"$\}$.*[7]

When $v = r_c$, the local configuration (r_c, A) is also called a local configuration in the component c.

[7] If c is a leaf component, then b_i is "ending" for each i. For technical reasons due to the exit vertex, we allow s_i to take the value of 0.

The *value* of a local configuration (v, A), denoted by $f(v, A)$, equals the minimum cost of a collection of $\ell(A)$ subtours in the subtree of c rooted at v, each subtour starting and ending at v, that together visit all of the terminals of the subtree of c rooted at v, where the i-th subtour visits s_i terminals and that $b_i =$ "passing" if and only if the i-th subtour visits e_c.

Let v be any vertex in c. We compute the function $f(v, \cdot)$ according to one of the three cases.

Case 1: v is the exit vertex e_c of the component c

For each $\ell \in [0, 2\Gamma/\alpha + 1]$, letting A denote the list consisting uniquely of ℓ identical pairs of $(0,\text{"passing"})$, we set $f(v, A) = 0$; for the remaining lists A, we set $f(v, A) = +\infty$.

Case 2: v is a leaf vertex of the tree $\hat{T}$

From Section 4 and the construction of $\hat{T}$, the leaf vertices in $\hat{T}$ are the same as the terminals in $\hat{T}$. Thus v is a terminal in $\hat{T}$. For the list A consisting of a single pair of $(1,\text{"ending"})$, we set $f(v, A) = 0$; for the remaining lists A, we set $f(v, A) = +\infty$.

Case 3: v is a non-leaf vertex of the tree $\hat{T}$ and $v \neq e_c$

Let v_1 and v_2 be the two children of v. We say that the local configurations (v_1, A_1), (v_2, A_2), and (v, A) are *compatible* if there is a partition $\mathcal{P}$ of $A_1 \cup A_2$ into parts, each part consisting of one or two pairs, and a one-to-one correspondence between every part in $\mathcal{P}$ and every pair in A such that:

- a part in $\mathcal{P}$ consisting of one pair $(s^{(1)}, b^{(1)})$ corresponds to a pair (s, b) in A if and only if $s^{(1)} = s$ and $b^{(1)} = b$;
- a part in $\mathcal{P}$ consisting of two pairs $(s^{(1)}, b^{(1)})$ and $(s^{(2)}, b^{(2)})$ corresponds to a pair (s, b) in A if and only if $s = s^{(1)} + s^{(2)}$ and

$$b = \begin{cases} \text{"passing"}, & \text{if } b^{(1)} \text{ is "passing" or } b^{(2)} \text{ is "passing"}, \\ \text{"ending"}, & \text{if } b^{(1)} \text{ is "ending" and } b^{(2)} \text{ is "ending"}. \end{cases}$$

We set

$$f(v, A) = \min \left\{ f(v_1, A_1) + f(v_2, A_2) + 2 \cdot \ell(A_1) \cdot w(v, v_1) + 2 \cdot \ell(A_2) \cdot w(v, v_2) \right\},$$

where the minimum is taken over all local configurations (v_1, A_1) and (v_2, A_2) that are compatible with (v, A).

The algorithm is very simple. See Algorithm 2.

Running time For each vertex v, since $\ell(A) = O_\epsilon(1)$, the number of local configurations (v, A) is $n^{O_\epsilon(1)}$. For fixed $(v_1, A_1), (v_2, A_2)$, and (v, A), there are $O_\epsilon(1)$ partitions of $A_1 \cup A_2$ into parts, so checking compatibility takes time $O_\epsilon(1)$. Thus the running time to compute the values at local configurations in a component c is $n^{O_\epsilon(1)}$. Since the number of components $c \in \mathcal{C}$ is at most n, the overall running time to compute the local configurations in all components $c \in \mathcal{C}$ is $n^{O_\epsilon(1)}$.

Algorithm 2 Computation for local configurations in a component c.

for each vertex $v \in c$ and each list A **do**
    $f(v, A) = +\infty$.
for each leaf vertex v of c **do** ▷ Cases 1 & 2
    Initialize $f(v, \cdot)$
for each non-leaf vertex v of c in the bottom-up order **do** ▷ Case 3
    Let v_1 and v_2 denote the two children of v
    for each local configurations (v_1, A_1), (v_2, A_2), and (v, A) **do**
        if (v_1, A_1), (v_2, A_2), and (v, A) are compatible **then**
            $f(v, A) \leftarrow \min(f(v, A), f(v_1, A_1) + f(v_2, A_2) + 2 \cdot \ell(A_1) \cdot w(v, v_1) + 2 \cdot \ell(A_2) \cdot w(v, v_2))$
return $f(r_c, \cdot)$

8.2 Subtree Configurations

In this subsection, we combine local configurations in the bottom up order to obtain *subtree configurations* (Definition 8.2), which are solutions restricted to subtrees of $\hat{T}$. The number of subtree configurations is polynomially bounded. When the subtree equals the tree $\hat{T}$, we obtain the entire solution to the CVRP.

Definition 8.2. *A* subtree configuration (v, A) *is defined by a vertex v and a list A consisting of $\ell(A)$ pairs $(\tilde{s}_1, n_1)$, $(\tilde{s}_2, n_2), \ldots, (\tilde{s}_\ell, n_{\ell(A)})$ such that*

- *v belongs to the set I (defined in Section 7.1); in other words, v is either the root of a component or a critical vertex;*
- $\ell(A) \leq \left(\frac{1}{\beta}\right)^{\frac{1}{\alpha}} + \frac{2\Gamma}{\alpha} + 1$ *if v is the root of a component, and* $\ell(A) \leq \left(\frac{1}{\beta}\right)^{\frac{1}{\alpha}}$ *if v is a critical vertex;*
- *for each $i \in [1, \ell(A)]$, $\tilde{s}_i$ is an integer in $[0, k]$ and n_i is an integer in $[0, n]$.*

The *value* of the subtree configuration (v, A), denoted by $g(v, A)$, is the minimum cost of a collection of $\ell(A)$ subtours in the subtree of $\hat{T}$ rooted at v, each subtour starting and ending at v, that together visit all of the real terminals of the subtree rooted at v, such that n_i subtours visit $\tilde{s}_i$ (real and dummy) terminals each.

To compute the values of subtree configurations, we consider the vertices $v \in I$ in the bottom up order. See Fig. 4 (Page 5). For each vertex $v \in I$ that is the root of a component, we compute the values $g(v, \cdot)$ using the algorithm in Section 8.2.1; and for each vertex $v \in I$ that is a critical vertex, we compute the values $g(v, \cdot)$ using the algorithm in Section 8.2.2.

8.2.1 Subtree Configurations at the Root of a Component

In this subsection, we compute the values of the subtree configurations at the root r_c of a component c. From Section 8.1, we have already computed the values of the local configurations in the component c.

If c is a leaf component, the local configurations in c induce the subtree configurations at r_c, in which $\tilde{s}_i = s_i$ and $n_i = 1$ for all i. Thus we obtain the values of the subtree configurations at r_c.

In the following, we consider the case when c is an internal component. We observe that the exit vertex e_c of the component c is a critical vertex. Thus the values of subtree configurations at e_c have already been computed using Algorithm 4 in Section 8.2.2 according to the bottom up order of the computation. To compute the value of a subtree configuration at r_c, we combine a subtree configuration at e_c and a local configuration in c, in the following way.

Consider a subtree configuration (e_c, A_e) and a local configuration (r_c, A_c), where

$$A_e = ((\tilde{s}_1, n_1), (\tilde{s}_2, n_2), \ldots, (\tilde{s}_{\ell_e}, n_{\ell_e})),$$

$$A_c = ((s_1, b_1), (s_2, b_2), \ldots, (s_{\ell_c}, b_{\ell_c})).$$

To each $i \in [1, \ell_c]$ such that b_i is "passing", we associate s_i with $\tilde{s}_j$ for some $j \in [1, \ell_e]$ with the constraints that $s_i + \tilde{s}_j \leq k$ (to guarantee that when we combine the two subtours, the result respects the capacity constraint) and that for each $j \in [1, \ell_e]$ at most n_j elements are associated to $\tilde{s}_j$ (because in the subtree rooted at e_c we only have n_j subtours of demand $\tilde{s}_j$ at our disposal). As a result, we obtain the list A of a subtree configuration (r_c, A) as follows:

- For each association $(s_i, \tilde{s}_j)$, we put in A the pair $(s_i + \tilde{s}_j, 1)$.
- For each pair $(\tilde{s}_j, n_j) \in A_e$, we put in A the pair $(\tilde{s}_j, n_j -$ (number of s_i's associated to $\tilde{s}_j$)).
- For each pair (s_i, "ending") $\in A_c$, we put in A the pair $(s_i, 1)$.

From the construction, $\ell(A) \leq \ell(A_e) + \ell(A_c)$. Since e_c is a critical vertex, $\ell(A_e) \leq \left(\frac{1}{\beta}\right)^{\frac{1}{\alpha}}$ by Definition 8.2. From Definition 8.1, $\ell(A_c) \leq \frac{2\Gamma}{\alpha} + 1$. Thus $\ell(A) \leq \left(\frac{1}{\beta}\right)^{\frac{1}{\alpha}} + \frac{2\Gamma}{\alpha} + 1$ as claimed in Definition 8.2.

Next, we compute the cost of the combination of (e_c, A_e) and (r_c, A_c); let x denote this cost. For any subtour t at e_c that is not associated to any non-spine passing subtour in the component c, we pay an extra cost to include the spine subtour of the component c, which is combined with the subtour t. The number of times that we include the spine subtour of c is the number of subtours at e_c minus the number of passing subtours in A_c, which is $\sum_{j \leq \ell_e} n_j - \sum_{i \leq \ell_c} \mathbb{1}[b_i \text{ is "passing"}]$. Thus we have

$$x = f(r_c, A_c) + g(e_c, A_e) + \mathrm{cost}(\mathrm{spine}_c) \cdot \left(\left(\sum_{j \leq \ell_e} n_j \right) - \left(\sum_{i \leq \ell_c} \mathbb{1}[b_i \text{ is "passing"}] \right) \right). \tag{5}$$

The algorithm is described in Algorithm 3.

Algorithm 3 Computation for subtree configurations at the root of a component c.

for each list A **do**
  $g(r_c, A) = +\infty$.
for each subtree configuration (e_c, A_e) and each local configuration (r_c, A_c) **do**
  for each way to combine (e_c, A_e) and (r_c, A_c) **do**
    $A \leftarrow$ the resulting list
    $x \leftarrow$ the cost computed in Equation (5)
    $g(r_c, A) \leftarrow \min(g(r_c, A), x)$.
return $g(r_c, \cdot)$

Running time The number of subtree configurations (e_c, A_e) and the number of local configurations (r_c, A_c) are both $n^{O_\epsilon(1)}$. For fixed (e_c, A_e) and (r_c, A_c), the number of ways to combine them is $O_\epsilon(1)$. Thus the running time of the algorithm is $n^{O_\epsilon(1)}$.

8.2.2 Subtree Configurations at a Critical Vertex

In this subsection, we compute the values of the subtree configurations at a critical vertex.

Let z denote any critical vertex. By Property 2 in Theorem 7.1, there exists a set X of $\frac{1}{\beta}$ integer values in $[\alpha \cdot k, k]$ such that the demands of the subtours in OPT_3 at the children of z are among the values in X. Thus the demand of a subtour at z in OPT_3 is the sum of at most $\frac{1}{\alpha}$ values in X. Therefore, the number of distinct demands of the subtours at z in OPT_3 is at most $(\frac{1}{\beta})^{\frac{1}{\alpha}} = O_\epsilon(1)$.

To compute a solution satisfying Property 2 in Theorem 7.1, a difficulty arises since the set X is unknown. Our approach is to enumerate all sets X of $\frac{1}{\beta}$ integer values in $[\alpha \cdot k, k]$, compute a solution with respect to each set X, and return the best solution found. Unless explicitly mentioned, we assume in the following that the set X is fixed.

Definition 8.3 (sum list)**.** *A sum list A consists of $\ell(A)$ pairs (s_1, n_1), (s_2, n_2), $\ldots$, $(s_{\ell(A)}, n_{\ell(A)})$ such that*

1. *$\ell(A) \leq (\frac{1}{\beta})^{\frac{1}{\alpha}}$;*
2. *For each $i \in [1, \ell(A)]$, $s_i \in [\alpha \cdot k, k]$ is the sum of a multiset of values in X and n_i is an integers in $[0, n]$.*

We require that in any subtree configuration (z, A), the list A is a sum list.

Let $r_1, r_2, \ldots, r_m$ be the children of z. For each $i \in [1, m]$, let

$$A_i = ((s_1^{(i)}, n_1^{(i)}), (s_2^{(i)}, n_2^{(i)}), \ldots, (s_{\ell_i}^{(i)}, n_{\ell_i}^{(i)}))$$

denote the list in a subtree configuration (r_i, A_i). We *round* the list A_i to a list

$$\overline{A_i} = ((\overline{s_1^{(i)}}, n_1^{(i)}), (\overline{s_2^{(i)}}, n_2^{(i)}), \ldots, (\overline{s_{\ell_i}^{(i)}}, n_{\ell_i}^{(i)})),$$

where $\overline{x}$ denotes the smallest value in X that is greater than or equal to x, for any integer value x. The rounding is represented by adding $\overline{x} - x$ *dummy* terminals at vertex r_i to each subtour initially consisting of x terminals. Let $\mathcal{S} \subseteq [1, k]$ denote a multiset such that for each $i \in [1, m]$ and for each $j \in [1, \ell_i]$, the multiset $\mathcal{S}$ contains $n_j^{(i)}$ copies of $\overline{s_j^{(i)}}$.

Definition 8.4 (compatibility)**.** *A multiset $\mathcal{S} \subseteq [1, k]$ and a sum list $((s_1, n_1), (s_2, n_2), \ldots, (s_\ell, n_\ell))$ are* compatible *if there is a partition of $\mathcal{S}$ into $\sum_i n_i$ parts and a correspondence between the parts of the partition and the values s_i's, such that for each s_i, there are n_i associated parts, and for each of those parts, the elements in that part sum up to s_i.*

Let $A = ((s_1, n_1), (s_2, n_2), \ldots, (s_{\ell(A)}, n_{\ell(A)}))$ be a sum list. The value $g(z, A)$ of the subtree configuration (z, A) equals the minimum, over all sets X and all subtree configurations $\{(r_i, A_i)\}_{1 \leq i \leq m}$ such that $\mathcal{S}$ and A are compatible, of

$$\sum_{i=1}^{m} g(r_i, A_i) + 2 \cdot n(A_i) \cdot w(r_i, z), \tag{6}$$

where $n(A_i)$ denotes $\sum_j n_j^{(i)}$. We note that $n_1 s_1 + n_2 s_2 + \cdots + n_{\ell(A)} s_{\ell(A)}$ is equal to the number of (real and dummy) terminals in the subtree rooted at z.

Fix any set X of $\frac{1}{\beta}$ integer values in $[\alpha \cdot k, k]$. We show how to compute the minimum cost of Eq. (6) over all subtree configurations $\{(r_i, A_i)\}_{1 \leq i \leq m}$ such that $\mathcal{S}$ and A are compatible. For each $i \in [1, m]$ and for each subtree configuration (r_i, A_i), the value $g(r_i, A_i)$ has already been computed

Algorithm 4 Computation for subtree configurations at a critical vertex z.

```
for each list A do
    g(z, A) ← +∞
for each set X of 1/β integer values in [α · k, k] do
    for each i ∈ [0, m] and each list A do
        DP_i(A) ← +∞
    DP_0(∅) ← 0
    for each i ∈ [1, m] do
        for each subtree configuration (r_i, A_i) do
            \overline{A_i} ← round(A_i)
            for each sum list A_{≤i−1} do
                for each way to combine A_{≤i−1} and \overline{A_i} do
                    A_{≤i} ← the resulting sum list
                    x ← DP_{i−1}(A_{≤i−1}) + g(r_i, A_i) + 2 · n(A_i) · w(r_i, z)
                    DP_i(A_{≤i}) ← min(DP_i(A_{≤i}), x)
    for each list A do
        g(z, A) ← min(g(z, A), DP_m(A))
return g(z, ·)
```

using the algorithm in Section 8.2.1, according to the bottom up order of the computation. We use a dynamic program that scans $r_1, \ldots, r_m$ one by one: those are all siblings, so here the reasoning is not bottom-up but left-right. Fix any $i \in [1, m]$. Let $\mathcal{S}_i \subseteq [1, k]$ denote a multiset such that for each $i' \in [1, i]$ and for each $j \in [1, \ell_{i'}]$, the multiset $\mathcal{S}_i$ contains $n_j^{(i')}$ copies of $\overline{s_j^{(i')}}$. We define a dynamic program table DP_i. The value $\mathrm{DP}_i(A_{\leq i})$ at a sum list $A_{\leq i}$ equals the minimum, over all subtree configurations $\{(r_{i'}, A_{i'})\}_{1 \leq i' \leq i}$ such that $\mathcal{S}_i$ and $A_{\leq i}$ are compatible, of

$$\sum_{i'=1}^{i} g(r_{i'}, A_{i'}) + 2 \cdot n(A_{i'}) \cdot w(r_{i'}, z).$$

When $i = m$, the values $\mathrm{DP}_m(\cdot)$ are those that we are looking for. It suffices to fill in the tables $\mathrm{DP}_1, \ldots, \mathrm{DP}_m$.

To compute the value DP_i at a sum list $A_{\leq i}$, we use the value DP_{i-1} at a sum list $A_{\leq i-1}$ and the value $g(r_i, A_i)$ of a subtree configuration (r_i, A_i). Let $A_{\leq i-1} = ((\hat{s}_1, \hat{n}_1), (\hat{s}_2, \hat{n}_2), \ldots, (\hat{s}_\ell, \hat{n}_\ell))$. We combine $A_{\leq i-1}$ and $\overline{A_i}$ as follows. For each $p \in [1, \ell]$ and each $j \in [1, \ell_i]$ such that $\hat{s}_p + \overline{s_j^{(i)}} \leq k$, we observe that $\hat{s}_p + \overline{s_j^{(i)}}$ is the sum of a multiset of values in X. We create $n_{p,j}$ copies of the association of $(\hat{s}_p, \overline{s_j^{(i)}})$, where $n_{p,j} \in [0, n]$ is an integer variable that we enumerate in the algorithm. We require that for each $p \in [1, \ell]$, $\sum_j n_{p,j} \leq \hat{n}_p$; and for each $j \in [1, \ell_i]$, $\sum_p n_{p,j} \leq n_j^{(i)}$. The resulting sum list $A_{\leq i}$ is obtained as follows.

- For each association $(\hat{s}_p, \overline{s_j^{(i)}})$, we put in $A_{\leq i}$ the pair $(\hat{s}_p + \overline{s_j^{(i)}}, n_{p,j})$.
- For each pair $(\hat{s}_p, \hat{n}_p) \in A_{\leq i-1}$, we put in $A_{\leq i}$ the pair $(\hat{s}_p, \hat{n}_p - \sum_j n_{p,j})$.
- For each pair $(\overline{s_j^{(i)}}, n_j^{(i)}) \in \overline{A_i}$, we put in $A_{\leq i}$ the pair $(\overline{s_j^{(i)}}, n_j^{(i)} - \sum_p n_{p,j})$.

The algorithm is described in Algorithm 4.

Running time Since the numbers of sets X, of subtree configurations, of sum lists, and of ways to combine them, are each $n^{O_\epsilon(1)}$, the running time of the algorithm is $n^{O_\epsilon(1)}$.

9 Reduction to Bounded Distances

In this section, we prove Theorem 2.3. We reduce the tree CVRP with general distances to the tree CVRP with bounded distances. The reduction holds for the unit demand version, the splittable version, and the unsplittable version of the tree CVRP.

9.1 Algorithm

For any subset $U \subseteq V'$ of terminals, a *subproblem on* U is an instance to the tree CVRP, in which the tree is $T = (V, E)$ and the set of terminals is U. To simplify the presentation, we assume that $1/\epsilon$ is an integer. For each integer $i \in \mathbb{Z}$, let $U_i \subseteq V'$ denote the set of terminals $v \in V'$ such that $\text{dist}(v) \in [(1/\epsilon)^i, (1/\epsilon)^{i+1})$.

Choose an integer $i_0 \in [0, (1/\epsilon) - 1]$ uniformly at random. For each integer $j \in \mathbb{Z}$, let $Y_j = U_{(1/\epsilon)\cdot j + i_0}$, and let $Z_j \subseteq V'$ denote the union of U_i for $i = (1/\epsilon)\cdot j + i_0 + 1, (1/\epsilon)\cdot j + i_0 + 2, \dots, (1/\epsilon)\cdot (j+1) + i_0 - 1$. Let W denote the collection of the non-empty sets Y_j's and the non-empty sets Z_j's. Note that W is a partition of the terminals in V'.

Let $\mathcal{A}$ denote any polynomial time ρ-approximation algorithm for the tree CVRP with bounded distances from the assumption in Theorem 2.3. Consider any set $U \in W$. From the construction, we have

$$\frac{\max_{v\in U} \text{dist}(v)}{\min_{v\in U} \text{dist}(v)} < \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1}.$$

Thus the subproblem on U has bounded distances. We apply the algorithm $\mathcal{A}$ on the subproblem on U to obtain a solution $\text{SOL}(U)$.

Let $\text{SOL} = \bigcup_{U\in W} \text{SOL}(U)$.

It is easy to see that SOL is a feasible solution to the CVRP. Since the number of subproblems is at most n and each subproblem is solved in polynomial time by $\mathcal{A}$, the overall running time is polynomial.

Remark 9.1. *The algorithm can be derandomized by enumerating all of the $1/\epsilon$ values of i_0, and returning the best solution found.*

9.2 Analysis

We analyze the cost of the solution SOL.

For any subset $U \subseteq V'$ of terminals, let $\text{opt}(U)$ denote the optimal value for the subproblem on U. For each set $U \in W$, $\text{SOL}(U)$ is a ρ-approximate solution to the subproblem on U. Thus we have

$$\text{cost}(\text{SOL}) = \sum_{U\in W} \text{cost}(\text{SOL}(U)) \le \rho \cdot \sum_{U\in W} \text{opt}(U) = \rho \cdot \left(\sum_j \text{opt}(Y_j) + \sum_j \text{opt}(Z_j)\right). \tag{7}$$

In the following, we bound $\sum_j \text{opt}(Y_j)$ and $\sum_j \text{opt}(Z_j)$.

For each $i \in \mathbb{Z}$, define $F_i \subseteq E$ to be

$$F_i = \left\{(u,v) \in E \mid \min(\text{dist}(u), \text{dist}(v)) \in \left[(1/\epsilon)^i, (1/\epsilon)^{i+1}\right) \right\}.$$

We assume without loss of generality that, for each $i \in \mathbb{Z}$ and for each $(u,v) \in F_i$, $\max(\mathrm{dist}(u), \mathrm{dist}(v)) \leq (1/\epsilon)^{i+1}$. Indeed, if $\max(\mathrm{dist}(u), \mathrm{dist}(v)) > (1/\epsilon)^{i+1}$, we may replace the edge (u,v) by a path of edges whose total weight equals $w(u,v)$, and such that each edge on that path satisfies the assumption.

Let $F_{\leq i}$ denote the union of $F_{i'}$ for all $i' \leq i$.

Lemma 9.2. $\sum_i \mathrm{opt}(U_i) \leq 2(1+\epsilon) \cdot \mathrm{opt}$.

Proof. Consider any tour t in OPT. Let i^* be the maximum index i such that $t \cap U_i \neq \emptyset$. Let t_{i^*} denote the tour obtained by pruning t so that it visits only the root and the terminals in U_{i^*}. Let $t_{\leq i^*-1}$ denote the tour obtained by pruning t so that it visits only the root and the remaining terminals, i.e., the terminals in U_i for all $i \leq i^*-1$. Let t' denote the common part of t_{i^*} and $t_{\leq i^*-1}$. We duplicate t' and charge the cost of t' to $t \cap F_{i^*-1}$. We have

$$\mathrm{cost}(t') \leq \mathrm{cost}(t_{i^*} \cap F_{\leq i^*-1}) \leq \frac{1}{1-\epsilon} \cdot \mathrm{cost}(t_{i^*} \cap F_{i^*-1}) \leq \frac{1}{1-\epsilon} \cdot \mathrm{cost}(t \cap F_{i^*-1}),$$

where the second inequality follows from the definition of $\{F_i\}_i$ and using the tree structure. We repeat on $t_{\leq i^*-1}$. We end up with a collection of tours, each visiting only terminals in the same set U_i for some i. Since the charges are to disjoint parts of t, the overall cost of the duplicated parts is at most $\frac{1}{1-\epsilon} \cdot \mathrm{cost}(t) < (1+2\epsilon) \cdot \mathrm{cost}(t)$. Summing over all tours in OPT concludes the proof. □

Using Lemma 9.2 and since $i_0 \in [0, (1/\epsilon) - 1]$ is chosen uniformly at random, we have

$$\mathbb{E}\Big[\sum_j \mathrm{opt}(Y_j)\Big] = \epsilon \cdot \sum_i \mathrm{opt}(U_i) \leq \epsilon \cdot 2(1+\epsilon) \cdot \mathrm{opt}. \tag{8}$$

Next, we analyze $\sum_j \mathrm{opt}(Z_j)$. Consider any tour t in OPT. First prune t so that it does not visit terminals in the set Y_j for any j. The rest of the analysis is similar to the proof of Lemma 9.2. Let j^* be the maximum index j such that $t \cap Z_j \neq \emptyset$. Let t_{j^*} denote the tour obtained by pruning t so that it visits only the root and the terminals in Z_{j^*}. Let $t_{\leq j^*-1}$ denote the tour obtained by pruning t so that it visits only the root and the remaining terminals, i.e, the terminals in Z_j for all $j \leq j^*-1$. Let t' denote the common part of t_{j^*} and $t_{\leq j^*-1}$. We duplicate t' and we charge the cost of t' to $t \cap F_{(1/\epsilon)\cdot j^* + i_0}$. We have

$$\mathrm{cost}(t') \leq \mathrm{cost}(t_{j^*} \cap F_{\leq (1/\epsilon)\cdot j^*+i_0-1}) \leq \frac{\epsilon}{1-\epsilon} \cdot \mathrm{cost}(t_{j^*} \cap F_{(1/\epsilon)\cdot j^*+i_0}) \leq \frac{\epsilon}{1-\epsilon} \cdot \mathrm{cost}(t \cap F_{(1/\epsilon)\cdot j^*+i_0}),$$

where the second inequality follows from the definition of $\{F_i\}_i$ and using the tree structure. We repeat on $t_{\leq j^*-1}$. We end up with a collection of tours, each visiting only terminals in the same set Z_j for some j. Since the charges are to disjoint parts of t, the overall cost of the duplicated parts is at most $\frac{\epsilon}{1-\epsilon} \cdot \mathrm{cost}(t) < 2\epsilon \cdot \mathrm{cost}(t)$. Summing over all tours in OPT, we have

$$\sum_j \mathrm{opt}(Z_j) < (1+2\epsilon) \cdot \mathrm{opt}. \tag{9}$$

From Eqs. (7) to (9), we have

$$\mathbb{E}\big[\mathrm{cost}(\mathrm{SOL})\big] \leq \rho \cdot (\epsilon \cdot 2(1+\epsilon) + (1+2\epsilon)) \cdot \mathrm{opt} < (1+5\epsilon)\rho \cdot \mathrm{opt}.$$

This completes the proof of Theorem 2.3.

10 Extension to the Splittable Tree CVRP

In this section, we prove Corollary 1.2 by extending the PTAS in Theorem 1.1 to the splittable setting.

Definition 10.1 (splittable tree CVRP)**.** *An instance of the* splittable *version of the* capacitated vehicle routing problem (CVRP) *on* trees *consists of*

- *an edge weighted* tree $T = (V, E)$ *with* $n = |V|$ *and with* root $r \in V$ *representing the* depot*,*
- *a set* $V' \subseteq V$ *of* terminals*,*
- *a positive integer* demand $d(v)$ *of each terminal* $v \in V'$*,*
- *a positive integer* tour capacity k*.*

A feasible solution is a set of tours such that

- *each tour starts and ends at* r*,*
- *each tour visits at most* k *demand,*
- *the demand of each terminal is covered, where we allow the demand of a terminal to be covered by multiple tours.*

The goal is to find a feasible solution such that the total cost of the tours is minimum.

We use a reduction from the splittable tree CVRP to the unit demand tree CVRP. The reduction was introduced by Jayaprakash and Salavatipour [JS22], which we summarize. First, we reduce an instance of the splittable tree CVRP to another instance of the splittable tree CVRP in which $d(v) \leq k \cdot n$ for any terminal v. Next, we replace each terminal v of T by a complete binary tree $T(v)$ of $d(v)$ leaves, such that each leaf of $T(v)$ is a terminal, and each edge of $T(v)$ has weight 0. Let $\tilde{T}$ denote the resulting tree. From the construction, $\tilde{T}$ contains at most $k \cdot n^2$ vertices. As observed in [JS22], the unit-demand CVRP on $\tilde{T}$ is equivalent to the splittable CVRP on T.

From Theorem 1.1, there is an approximation scheme for the unit demand tree CVRP with running time polynomial in the number of vertices. Therefore, we obtain an approximation scheme for the splittable tree CVRP with running time polynomial in n and k.

References

[ACL10] Anna Adamaszek, Artur Czumaj, and Andrzej Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. *International Journal of Foundations of Computer Science*, 21(06):893–904, 2010.

[AG90] Kemal Altinkemer and Bezalel Gavish. Heuristics for delivery problems with constant error guarantees. *Transportation Science*, 24(4):294–297, 1990.

[AGM16] S. P. Anbuudayasankar, K. Ganesh, and Sanjay Mohapatra. *Models for practical routing problems in logistics*. Springer, 2016.

[AKK01] Tetsuo Asano, Naoki Katoh, and Kazuhiro Kawashima. A new approximation algorithm for the capacitated vehicle routing problem on a tree. *Journal of Combinatorial Optimization*, 5(2):213–231, 2001.

[AKTT97] Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi 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 Theory of computing*, pages 275–283, 1997.

[BDO06] Agustín Bompadre, Moshe Dror, and James B. Orlin. Improved bounds for vehicle routing solutions. *Discrete Optimization*, 3(4):299–316, 2006.

[Bec18] Amariah Becker. A tight 4/3 approximation for capacitated vehicle routing in trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 116 of *LIPIcs*, pages 3:1–3:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BKS17] Amariah Becker, Philip N. Klein, and David 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)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[BKS18] Amariah Becker, Philip N. Klein, and David Saulpic. 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)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[BKS19] Amariah Becker, Philip N. Klein, and Aaron Schild. A PTAS for bounded-capacity vehicle routing in planar graphs. In *Workshop on Algorithms and Data Structures*, pages 99–111. Springer, 2019.

[BP19] Amariah Becker and Alice Paul. A framework for vehicle routing approximation schemes in trees. In *Workshop on Algorithms and Data Structures*, pages 112–125. Springer, 2019.

[BTV21] Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 1–14. Springer, 2021.

[CFKL20] Vincent Cohen-Addad, Arnold Filtser, Philip N. Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 589–600. IEEE, 2020.

[CL12] Teodor G. Crainic and Gilbert Laporte. *Fleet management and logistics*. Springer Science & Business Media, 2012.

[DM15] Aparna Das and Claire Mathieu. A quasipolynomial time approximation scheme for Euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015.

[DR59] George B. Dantzig and John H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[FMRS21] Zachary Friggstad, Ramin Mousavi, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Improved approximations for CVRP with unsplittable demands. *arXiv preprint arXiv:2111.08138*, 2021.

[GRW08] Bruce Golden, S. Raghavan, and Edward Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*. Springer, 2008.

[GW81] Bruce L. Golden and Richard T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

[HK98] Shin-ya Hamaguchi and Naoki Katoh. A capacitated vehicle routing problem on a tree. In *International Symposium on Algorithms and Computation*, pages 399–407. Springer, 1998.

[HR85] Mordecai Haimovich and Alexander H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

[JS22] Aditya Jayaprakash and Mohammad R. Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 877–893, 2022.

[KD16] Michael Khachay and Roman Dubinin. PTAS for the Euclidean capacitated vehicle routing problem in $\mathbb{R}^d$. In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer, 2016.

[LLM91] Martine Labbé, Gilbert Laporte, and Hélene Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991.

[MZ22] Claire Mathieu and Hang Zhou. A tight $(1.5 + \epsilon)$-approximation for unsplittable capacitated vehicle routing on trees. `https://arxiv.org/abs/2202.05691`, 2022.

[TV02] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, 2002.

A Decomposition of the Tree into Components

In this section, we prove Lemma 4.2.

A *leaf component* is a subtree rooted at a vertex $v \in V$ containing at least $\Gamma \cdot k$ terminals and such that each of the subtrees rooted at the children of v contains strictly less than $\Gamma \cdot k$ terminals. Observe that the leaf components are disjoint subtrees of T. The *backbone* of T is the partial subtree of T consisting of all edges on paths from the root of T to the root of some leaf component.

Definition A.1 (key vertices)**.** *We say that a vertex $v \in V$ is a* key vertex *if v is of one of the three cases: (1) the root of a leaf component; (2) a branch point of the backbone; (3) the root of the tree T.*

We say that two key vertices v_1, v_2 are *consecutive* if the v_1-to-v_2 path in the tree does not contain any other key vertex. For each pair of consecutive key vertices (v_1, v_2), we consider the subgraph *between* v_1 and v_2, and decompose that subgraph into *internal components*, each of demand at most $2\Gamma \cdot k$, such that all of these components are *big* (i.e., of demand at least $\Gamma \cdot k$) except for the upmost component.

A formal description of the construction is given in Algorithm 5.

Algorithm 5 Decomposition into components.

for each vertex $v \in V$ **do**
    $T(v) \leftarrow$ subtree of T rooted at v
    $n(v) \leftarrow$ number of terminals in $T(v)$
for each non-leaf vertex $v \in V$ **do**
    Let v_1 and v_2 denote the two children of v in T
    if $n(v) \geq \Gamma \cdot k$ and $n(v_1) < \Gamma \cdot k$ and $n(v_2) < \Gamma \cdot k$ **then**
        Let $T(v)$ be a leaf component with root vertex v

$B \leftarrow$ set of *key vertices* ▷ Definition A.1
for each vertex $v_2 \in B$ such that $v_2 \neq r$ **do**
    $v_1 \leftarrow$ lowest ancestor of v_2 among vertices in B ▷ v_1 and v_2 are *consecutive* key vertices
    for each vertex v on the v_1-to-v_2 path **do** $H(v) \leftarrow (T(v) \setminus T(v_2)) \cup \{v_2\}$
    $x \leftarrow v_2$
    while $H(v_1)$ contains at least $\Gamma \cdot k$ terminals **do**
        $v \leftarrow$ the deepest vertex on the v_1-to-x path such that $H(v)$ contains at least $\Gamma \cdot k$ terminals
        Let $H(v)$ be an internal component with root vertex v and exit vertex x
        $x \leftarrow v$
        for each vertex v' on the v_1-to-x path **do** $H(v') \leftarrow (T(v') \setminus T(x)) \cup \{x\}$
    if $v_1 \neq x$ **then**
        Let $H(v_1)$ be an internal component with root vertex v_1 and exit vertex x

The first three properties in the claim follow from the construction. It remains to show the last property in the claim. For each big component c, we define the image of c to be itself. It remains to consider the components that are not big, called *bad* components. Observe that the root vertex r_c of any bad component c is a key vertex. We say that a bad component c is a *left bad* component (resp. *right bad* component) if c contains the left child (resp. right child) of r_c. We define a map from left bad components to leaf components, such that the image of a left bad component is the leaf component that is *rightmost* among its descendants, and we show that this map is injective. Let c_1 and c_2 be any left bad components. Observe that r_{c_1} and r_{c_2} are distinct key vertices. If r_{c_1} is ancestor of

r_{c_2} (the case when r_{c_2} is ancestor of r_{c_1} is similar), then the image of c_2 is in the left subtree of r_{c_2} whereas the image of c_1 is outside the left subtree of r_{c_2}, so the images of c_1 and of c_2 are different. In the remaining case, the subtrees rooted at r_{c_1} and at r_{c_2} are disjoint, so the images of c_1 and of c_2 are different. Thus the map for left bad components is injective. Note that every leaf component is big. Therefore, we obtain an injective map from left bad components to big components such that the image of a left bad component is among its descendants. The map for right bad components is symmetric. Hence the first part of the last property. The second part of the last property follows from the first part of that property and the fact that the number of components with demands at least Γ is at most $1/\Gamma$ times the total demand in the tree T. This completes the proof of Lemma 4.2.