An Approximation Algorithm for Distance-Constrained Vehicle Routing on Trees

Marc Dufay
École Polytechnique and IRIF, Palaiseau, France

Claire Mathieu
CNRS Paris, France

Hang Zhou
École Polytechnique, Palaiseau, France

Abstract

In the Distance-constrained Vehicle Routing Problem (DVRP), we are given a graph with integer edge weights, a depot, a set of n terminals, and a distance constraint D. The goal is to find a minimum number of tours starting and ending at the depot such that those tours together cover all the terminals and the length of each tour is at most D.

The DVRP on *trees* is of independent interest, because it is equivalent to the "virtual machine packing" problem on trees studied by Sindelar et al. [SPAA'11]. We design a simple and natural approximation algorithm for the tree DVRP, parameterized by $\varepsilon > 0$. We show that its approximation ratio is $\alpha + \varepsilon$, where $\alpha \approx 1.691$, and in addition, that our analysis is essentially tight. The running time is polynomial in n and D. The approximation ratio improves on the ratio of 2 due to Nagarajan and Ravi [Networks'12].

The main novelty of this paper lies in the analysis of the algorithm. It relies on a reduction from the tree DVRP to the bounded space online bin packing problem via a new notion of "reduced length".

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases vehicle routing, distance constraint, approximation algorithms, trees

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.27

Funding This work was partially supported by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

1 Introduction

The vehicle routing problem is arguably one of the most important problems in Operations Research. Books have been dedicated to vehicle routing problems, e.g., [4, 7, 16]. Yet, these problems remain challenging, both from a practical and a theoretical perspective. As observed by Li, Simchi-Levi, and Desrochers [11]:

> Typically, two types of constraints are imposed on the route traveled by any vehicle. One is the *capacity constraint* in which each vehicle cannot serve more than a given number of customers. The second is the *distance constraint*: The total distance traveled by each vehicle should not exceed a prespecified number. Depending on the system, one or both types can be binding. Usually delivery and pick-up services are characterized by capacity constraints [...], while service systems are characterized by distance constraints (see, for example, [1]). On the latter case, the system is usually required to provide a visit of a skilled worker at customer sites and thus the length of routes must be controlled because these are related to working hours.

The focus of this paper is the distance constraint. In the *Distance-constrained Vehicle Routing Problem (DVRP)*, we are given a graph with integer edge weights, a vertex called *depot*, a set of n vertices called *terminals*, and an integer *distance constraint* D. The goal is to find a minimum number of tours starting and ending at the depot such that those tours together cover all the terminals and the length of each tour is at most D. Friggstad and Swamy [5] gave an $O(\frac{\log D}{\log\log D})$-approximation, improving upon an $O(\log D)$-approximation of Nagarajan and Ravi [13].[1] Experimental results were given in [9].

The DVRP on *trees* is of independent interest, because of its relation to the *Virtual Machine (VM) packing problem* [15, 2, 14]. In the VM packing problem, we are given a set of VMs that must be hosted on physical servers, where each VM consists of a set of pages and each physical server has a capacity of D pages. VMs running on the same physical server may share pages. The goal is to pack the VMs onto the smallest number of physical servers. Sindelar et al. [15] observed:

> Using memory traces for a mixture of diverse OSes, architectures, and software libraries, we find that *a tree model* can capture up to 67% of inter-VM sharing from these traces.

Sindelar et al. [15] gave a 3-approximation for the VM packing problem on trees, and also suggested as future work "*A key direction is tightening the approximation bounds*".

It is easy to see that the VM packing problem on trees is equivalent to the DVRP on trees. Thus the algorithm of Sindelar et al. [15] is a 3-approximation for the tree DVRP. Nagarajan and Ravi [13] improved the ratio of the tree DVRP to 2. When the distance bound is allowed to be violated by an ε fraction, Becker and Paul [3] designed a bicriteria PTAS. We observe that the tree DVRP is strongly NP-hard (Appendix A).

In this work, we design a simple and natural approximation algorithm for the tree DVRP, parameterized by $\epsilon > 0$, see Algorithm 1. The main novelty lies in the analysis of Algorithm 1. Our main result (Theorem 2) shows that the approximation ratio of Algorithm 1 is $\alpha + O(\epsilon)$, where $\alpha \approx 1.691$ is defined in Definition 1. The running time is polynomial in n and D. Interestingly, the ratio α is best possible for Algorithm 1 (Theorem 3).

▶ **Definition 1.** *Let* $(u_k)_{k\geq 1}$ *denote the following sequence:*

$$u_k = \begin{cases} 1, & k = 1, \\ u_{k-1}(u_{k-1}+1), & k \geq 2. \end{cases}$$

$$\text{Let } \alpha := \sum_{k=1}^{+\infty} \frac{1}{u_k} = 1.69103\ldots.$$

▶ **Theorem 2.** *For any constant* $\varepsilon > 0$*, Algorithm 1 is an* $(\alpha + O(\epsilon))$*-approximation for the* Distance-constrained Vehicle Routing Problem (DVRP) *on trees. Its running time is* $O\left(n^2 \cdot \left(D/\epsilon^2\right)^{O(1/\epsilon^2)}\right)$.

▶ Remark. It is common in vehicle routing to parameterize the running time of an algorithm by the value of a constraint. For example, in capacitated vehicle routing with splittable demands, the running time of the algorithms in [8, 12] is parameterized by the *tour capacity*.

[1] More precisely, Nagarajan and Ravi [13] designed a bicriteria $\left(O(\log \frac{1}{\varepsilon}), 1+\varepsilon\right)$-approximation, which could be turned into an $O(\log D)$-approximation by setting $D = \frac{1}{\varepsilon}$.

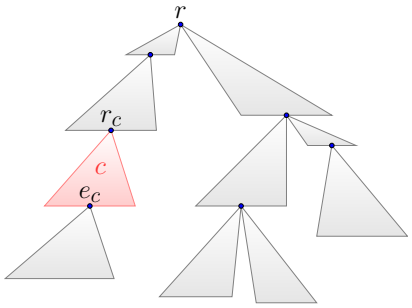


Figure 1 Component decomposition. Extracted from [12].

▶ **Theorem 3.** *For any constant $\varepsilon > 0$, Algorithm 1 is at best an α-approximation for the* Distance-constrained Vehicle Routing Problem (DVRP) *on trees.*

It is an open question to achieve a better-than-α approximation for the DVRP on trees. From Theorem 3, this would require new insights in the algorithmic design.

1.1 Algorithm

Let $\varepsilon > 0$. Our algorithm for the DVRP is Algorithm 1. It consists of two phases, using Algorithm 2 and Algorithm 3 with $\Gamma = 1/\varepsilon^2$:

Phase 1: The tree is partitioned into *components* (Lemma 5 and Algorithm 2), where each component can be covered with a bounded number of tours.

Phase 2: Each component is taken as an independent instance for the DVRP, which is solved using a polynomial time dynamic program (Lemma 7 and Algorithm 3); the solution for the whole tree is the union of the solutions for individual components.

Algorithm 1 Approximation algorithm for the DVRP on trees. Parameter $\varepsilon > 0$.

Input: A tree T rooted at r, a set of terminals U, a distance constraint D
Output: number of tours in a feasible solution to cover all terminals in U

1: $\Gamma \leftarrow 1/\epsilon^2$
2: Partition the tree T into a set $\mathcal{C}$ of components ▷ Algorithm 2
3: **for** each component $c \in \mathcal{C}$ **do**
4: $r_c \leftarrow$ root vertex of component c ▷ defined in Lemma 5
5: $D_c \leftarrow D$ minus twice the distance between r and r_c
6: $U_c \leftarrow$ set of terminals in U that belong to c
7: $n_c \leftarrow$ minimum number of tours for the subproblem (c, U_c, D_c) ▷ Algorithm 3
8: **return** $\sum_{c \in \mathcal{C}} n_c$

▶ Remark. As written, Algorithm 1 returns the number of tours, not the tours themselves. If desired, by adding auxiliary information to the dynamic program of Algorithm 3 in a standard manner, it is possible to retrieve a feasible solution whose number of tours matches the value returned by Algorithm 1.

1.2 Overview of the Analysis

This section gives a high-level description of main new ideas in this paper.

The analysis of Algorithm 1 relies on a reduction (Theorem 14) from the DVRP on trees to the *bounded space online bin packing* (Definition 12). In the *bounded space online bin packing*, the items arrive in an online fashion, and in addition, at any point in the packing process, only a fixed number of partially-filled bins may be *active*, i.e., open to further items. Once a bin is closed it must remain so and can no longer be active.

What does bin packing have to do with DVRP on trees? The relation lies in a new notion introduced in this paper, that of *reduced lengths* (Definition 8). If we consider a bin in bin packing, its item sizes sum to at most 1. Similarly, if we consider a tour in DVRP on trees, the reduced lengths of its subtours in components sum to at most 1 (Lemma 9).

To show the reduction (Theorem 14), we start from an instance of the tree DVRP and we construct an instance of the bounded space online bin packing as follows. We consider the tours in an optimal solution and decompose each tour into a set of subtours, one for each component it visits. For each subtour, we consider its *reduced length*, which is defined with respect to the component. We then construct an online sequence of those reduced lengths such that reduced lengths of the subtours in the same component are *consecutive* in the sequence. Then we consider a solution to the bounded space online bin packing on that sequence. Intuitively, the bound on the number of open bins implies that bins in that solution tend to contain only reduced lengths of the subtours from the same component. That is a desirable property because, if a bin contains only reduced lengths of subtours in the same component, then those subtours can be combined into a single tour (Lemma 10). Using the above intuition, we show that the performance of Algorithm 1 for the tree DVRP is up to some negligible additive factor at least as good as the best corresponding bounded space bin-packing problem.

From the reduction (Theorem 14) and since the bounded space online bin packing admits an algorithm with worst-case performance ratio α due to Lee and Lee [10], we conclude that Algorithm 1 is an $(\alpha + O(\varepsilon))$ approximation for the DVRP on trees (Theorem 2).

There are several technical details along the way. For example, subtours that end in a component and subtours that traverse a component cannot be combined in the same way, which requires additional care in the definition of reduced lengths, see Figure 2.

Finally, we show that the analysis in Theorem 2 on the approximation ratio of Algorithm 1 is essentially tight by providing a matching lower bound (Theorem 3).

1.3 Organization of the Paper

In Section 2, we give the formal problem definition, preliminary results, and previous techniques. In Section 3, we define and analyze *reduced lengths*. In Section 4, we prove Theorem 2 by establishing the reduction (Theorem 14). In Section 5, we prove Theorem 3.

2 Preliminaries

2.1 Formal Problem Definition, Notations, and Assumptions

Let T be a rooted tree (V, E) with non-negative integer edge weights $w(u, v)$ for all $(u, v) \in E$. Consider a tour (resp. subtour) $t = (v_0, v_1, v_2, \ldots, v_m)$ for some $m \in \mathbb{N}$ such that $v_0 = v_m$. The *length* of t, denoted by $\text{length}(t)$, is defined to be $\sum_{i=1}^{m} w(e_i)$, where e_i is the edge between v_{i-1} and v_i.

▶ **Definition 4** (tree DVRP). *An instance* (T, U, D) *of the* Distance-constrained Vehicle Routing Problem (DVRP) *on* trees *consists of*

- *a rooted* tree $T = (V, E)$ *with non-negative integer edge weights, where the root* $r \in V$ *of the tree is the* depot*;*
- *a set* $U \subseteq V$ *of* n *vertices of the tree* T*, called* terminals*;*
- *a positive integer* distance constraint D*.*

A feasible solution is a set of tours such that

- *each tour starts and ends at* r*;*
- *each terminal is visited by at least one tour;*
- *each tour has length at most* D*.*

The goal is to minimize the total number of tours in a feasible solution.

Let OPT denote an optimal solution to the tree DVRP. Let opt denote the number of tours in OPT.

For any vertices $u, v \in V$, let $\text{dist}(u, v)$ denote the *distance* between u and v in the tree T.

Up to a preprocessing step, we can assume that each vertex has at most two children and that the terminals are the same as the leaves of the tree, see, e.g., [12]. Furthermore, we assume that each non-leaf vertex has exactly two children.[2] We assume that for each tour in the solution, each edge on that tour is traversed exactly twice, once in each direction. We assume w.l.o.g. that $\varepsilon > 0$ is upper bounded by a sufficiently small constant.

2.2 Decomposition into Components

We decompose the tree into components in Lemma 5.

▶ **Lemma 5** ([12]). *Let* $\Gamma \geq 1$ *be a fixed integer. There is an algorithm* $\textsc{Decompose}^{(\Gamma)}$ *(Algorithm 2) that runs in time* $O(n^2 \cdot (2\Gamma)^\Gamma \cdot D^{3\Gamma})$ *and computes a partition of the edges of the tree* T *into a set* $\mathcal{C}$ *of* components *(see Figure 1), such that all of the following properties are satisfied:*

1. *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.*
2. *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 an* internal *component otherwise. An internal component* c *shares vertices with other components at two vertices only: at vertex* r_c*, and at one other vertex, called the* exit *vertex of the component* c*, and denoted by* e_c*.*
3. *The terminals of each component can be covered by at most* Γ *tours. We say that a component is* big *if at least* $\Gamma/2$ *tours are needed to cover its terminals. Each leaf component is big.*
4. *If the number of components in* $\mathcal{C}$ *is strictly greater than 1, then there exists a map from all components to big components, such that each big component has at most three pre-images.*

The component decomposition was previously given in [12], inspired by Becker and Paul [3]. Algorithm 2 is a small adaptation from [12], and is given in Appendix B. The proof of Lemma 5 is almost identical to that in [12], hence omitted.

[2] Indeed, if a vertex u has only one child v, there are two cases. In the first case, u is the root. Then we remove u and its incident edge, let v be the depot, and update D by $D - 2w(u, v)$. In the second case, u has a parent p. Then we remove u and its incident edges, and we add an edge between p and v with weight $w(p, v) := w(p, u) + w(u, v)$.

▶ **Definition 6** (subtours and their categories, [12]). *Let $c \in \mathcal{C}$ be any component. A* subtour *in component c is a walk inside c that starts and ends at r_c and visits at least one terminal. For any subtour s in component c, we say that the* category *of s is* passing *if c is an internal component and the exit vertex e_c belongs to s, and* ending *otherwise.*

2.3 Solving Instances with a Bounded Number of Tours

For an instance of the tree DVRP admitting a solution of a bounded number of tours, an optimal solution can be computed in polynomial time using a simple dynamic program (Algorithm 3), in Lemma 7.

▶ **Lemma 7.** *Let $\Gamma \geq 1$ be a fixed integer. There is an algorithm $\textsc{Solve}^{(\Gamma)}$ (Algorithm 3) that, given an instance $(\tilde{T}, \tilde{U}, \tilde{D})$ of the DVRP on trees, computes*

$$\min\{|S| : S \text{ is a feasible set of tours and } |S| \leq \Gamma\}.$$

Thus $\textsc{Solve}^{(\Gamma)}$ returns $+\infty$ if the instance does not admit any solution of at most Γ tours. The running time of $\textsc{Solve}^{(\Gamma)}$ is $O(\tilde{n} \cdot (2\Gamma)^{\Gamma} \cdot \tilde{D}^{3\Gamma})$, where $\tilde{n}$ is the number of terminals in $\tilde{T}$.

Algorithm 3 and the proof of Lemma 7 are in Appendix C.

3 Reduced Lengths

In this section, we introduce a novel concept of *reduced lengths* (Definition 8).

▶ **Definition 8** (reduced lengths, see Figure 2). *Let $c \in \mathcal{C}$ be any component. For any subtour s in component c, we define the* reduced length $\bar{\ell}(s)$ *of subtour s as follows:*

- *If s is an* ending subtour*,*

$$\bar{\ell}(s) := \frac{\text{length}(s)}{D - 2 \cdot \text{dist}(r, r_c)};$$

- *If s is a* passing subtour*,*

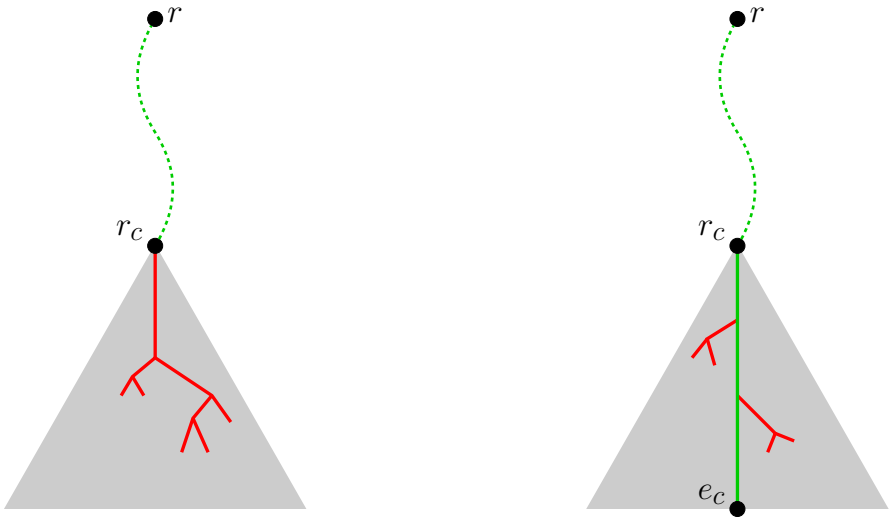
$$\bar{\ell}(s) := \frac{\text{length}(s) - 2 \cdot \text{dist}(r_c, e_c)}{D - 2 \cdot \text{dist}(r, e_c)}.$$

We distinguish the two categories of the subtours in the definition of reduced lengths (Definition 8) so that the properties in Lemmas 9 and 10 are satisfied.

▶ **Lemma 9.** *Let t be a tour. Let $c_1, \ldots, c_k$ be the components containing terminals visited by t. For each $i \in [1, k]$, let s_i denote the subtour of t in c_i. Then $\sum_{i=1}^{k} \bar{\ell}(s_i) \leq 1$.*

Proof. Let $c^* \in \{c_1, \ldots, c_k\}$ be a component such that $\text{dist}(r, r_{c^*})$ is maximized. Let p be the path from r to r_{c^*}. For each $i \in [1, k]$, let $E_i \subseteq E$ denote a subset of edges defined as follows: if s_i is an ending subtour, then E_i consists of the edges in s_i; and if s_i is a passing subtour, then E_i consists of the edges in s_i that do not belong to the r_{c_i}-to-e_{c_i} path. By construction, the edges in $p, E_1, \ldots, E_k$ are disjoint. Let $w(E_i)$ denote $\sum_{e \in E_i} w(e)$. Since the length of t is at most D, we have:

$$2 \cdot \text{dist}(r, r_{c^*}) + \sum_{i=1}^{k} 2 \cdot w(E_i) \leq D,$$



(a) Ending subtour s. **(b)** Passing subtour s.

Figure 2 Illustration for the new notion of the *reduced length* of a subtour s in a component c. The component c is represented by the gray triangle. There are two cases depending on whether s is an ending subtour (Figure 2a) or a passing subtour (Figure 2b). In both cases, the subtour s is represented by the solid segments. The r-to-r_c connection (in both directions) is represented by the dashed curve. The reduced length $\bar{\ell}(s)$ of the subtour s is defined by $\frac{\text{length(red)}}{D - \text{length(green)}}$, which is proportional to the red part of the path. See Definition 8.

and equivalently,

$$\sum_{i=1}^{k} \frac{2 \cdot w(E_i)}{D - 2 \cdot \text{dist}(r, r_{c^*})} \leq 1. \tag{1}$$

For each $i \in [1, k]$, by the definition of E_i, we have

$$2 \cdot w(E_i) = \begin{cases} \text{length}(s_i), & \text{if } s_i \text{ is an ending subtour} \\ \text{length}(s_i) - 2 \cdot \text{dist}(r_{c_i}, e_{c_i}), & \text{if } s_i \text{ is a passing subtour.} \end{cases} \tag{2}$$

By the choice of c^*, we have

$$\text{dist}(r, r_{c^*}) \geq \begin{cases} \text{dist}(r, r_{c_i}), & \text{if } s_i \text{ is an ending subtour} \\ \text{dist}(r, e_{c_i}), & \text{if } s_i \text{ is a passing subtour.} \end{cases} \tag{3}$$

From Equations (2) and (3) and Definition 8, for all $i \in [1, k]$, we have

$$\bar{\ell}(s_i) \leq \frac{2 \cdot w(E_i)}{D - 2 \cdot \text{dist}(r, r_{c^*})}.$$

Combining with Equation (1), the claim follows. ◀

▶ **Lemma 10.** *Let c be any component. Let $s_1, \ldots, s_m$, for some $m \geq 1$, be any subtours in c that are of the same category and such that $\sum_{i=1}^{m} \bar{\ell}(s_i) \leq 1$. Then all terminals from all subtours $s_1, \ldots, s_m$ can be visited by a tour of length at most D.*

Proof. We construct a tour t visiting all terminals from all subtours $s_1, \ldots, s_m$, and we show that its length is at most D. Since $s_1, \ldots, s_m$ are of the same category, there are two cases depending on their category.

- If $s_1, \ldots, s_m$ are ending subtours, then t consists of the r-to-r_c connection (in both directions) and of the subtour s_i for each $i \in [1, m]$. Therefore,

$$\begin{aligned}\text{length}(t) &\leq 2 \cdot \text{dist}(r, r_c) + \sum_{i=1}^{m} \text{length}(s_i) \\ &= 2 \cdot \text{dist}(r, r_c) + (D - 2 \cdot \text{dist}(r, r_c)) \sum_{i=1}^{m} \bar{\ell}(s_i) && \text{(by Definition 8)} \\ &\leq D && \text{(since } \sum_{i=1}^{m} \bar{\ell}(s_i) \leq 1\text{).}\end{aligned}$$

- If $s_1, \ldots, s_m$ are passing subtours, then t consists of the r-to-e_c connection (in both directions) and of the subtour s_i without the r_c-to-e_c connection (in both directions) for each $i \in [1, m]$. Therefore,

$$\begin{aligned}\text{length}(t) &\leq 2 \cdot \text{dist}(r, e_c) + \sum_{i=1}^{m} (\text{length}(s_i) - 2 \cdot \text{dist}(r_c, e_c)) \\ &= 2 \cdot \text{dist}(r, e_c) + (D - 2 \cdot \text{dist}(r, e_c)) \sum_{i=1}^{m} \bar{\ell}(s_i) && \text{(by Definition 8)} \\ &\leq D && \text{(since } \sum_{i=1}^{m} \bar{\ell}(s_i) \leq 1\text{).}\end{aligned}$$

The claim follows in both cases. ◀

▶ **Lemma 11.** *Assuming $\Gamma \geq 20$, the number of components in $\mathcal{C}$ is at most $(15/\Gamma) \cdot \text{opt}$.*

Proof. Let $\mathcal{C}_b \subseteq \mathcal{C}$ denote the set of big components in $\mathcal{C}$. Consider a big component $c \in \mathcal{C}_b$. Let $s_{c,1}, ..., s_{c,m_c}$ denote all subtours in c in the global optimal solution, for some $m_c \in \mathbb{N}$. Let $\bar{\ell}_{c,1}, \ldots, \bar{\ell}_{c,m_c}$ denote their reduced lengths. Those subtours can be viewed as a feasible solution to the local problem for the terminals in c only. We partition those subtours into parts, such that the subtours in the same part are of the same category, and in addition, for each part except possibly two parts, the reduced lengths of the subtours in that part sum to a value in $(1/2, 1]$. For each part, we replace the subtours by a new subtour visiting the terminals covered by all the subtours in that part. This creates a new feasible local solution for the terminals of c, and its number of subtours is at most $2\sum_{i=1}^{m_c} \bar{\ell}_{c,i} + 2$. By definition of big components, covering the terminals of c requires at least $\Gamma/2$ tours. Thus:

$$2\sum_{i=1}^{m_c} \bar{\ell}_{c,i} + 2 \geq \Gamma/2.$$

Therefore,

$$\sum_{i=1}^{m_c} \bar{\ell}_{c,i} \geq \Gamma/4 - 1 \geq \Gamma/5,$$

since $\Gamma \geq 20$. Hence

$$\sum_{c \in \mathcal{C}} \sum_{i=1}^{m_c} \bar{\ell}_{c,i} \geq |\mathcal{C}_b| \cdot \Gamma/5 \geq |\mathcal{C}| \cdot \Gamma/15,$$

where the last inequality follows from Property 4 of Lemma 5.

On the other hand, consider each tour t in OPT. Let k_t denote the number of components containing terminals visited by tour t. For each $i \in [1, k_t]$, let $\bar{\ell}'_{t,i}$ denote the reduced length of the i^{th} subtour of t. By Lemma 9 we have:

$$\sum_{t \in \text{OPT}} \sum_{i=1}^{k_t} \bar{\ell}'_{t,i} \leq \sum_{t \in \text{OPT}} 1 = \text{opt}.$$

By re-ordering this sum we get

$$\sum_{t \in \text{OPT}} \sum_{i=1}^{k_t} \bar{\ell}'_{t,i} = \sum_{c \in \mathcal{C}} \sum_{i=1}^{m_c} \bar{\ell}_{c,i}.$$

Therefore, $\text{opt} \geq |\mathcal{C}| \cdot \Gamma/15$. The claim follows. ◀

4 Proof of Theorem 2

To prove Theorem 2, we use a reduction from the DVRP on trees to the *bounded space online bin-packing*.

4.1 Bounded Space Online Bin Packing

▶ **Definition 12.** *In the* bounded space online bin-packing *problem, we are given a positive integer M and an online sequence of item sizes $(a_1, a_2, ..., a_n) \in [0, 1]^n$, and we want to pack those items into bins of size* 1*. At any time, the following operations are allowed: (1) opening a bin; (2) closing an bin; (3) assigning the current item to some* open *bin. We require that at any time there are at most M open bins.*

The goal is to minimize the total number of bins used.

Let opt_{BP} denote the number of bins in an optimal solution to the bin packing in the *offline* setting. The following theorem due to Lee and Lee [10] is a direct corollary of Theorem 2 from [10] and Eq. (3.6) from [10].

▶ **Theorem 13** ([10])**.** *Let M be any positive integer. Let $k \in \mathbb{N}$ be such that $u_k < M \leq u_{k+1}$. There exists a solution to the bounded space online bin-packing*

in which the total number of bins used is at most

$$\left(\sum_{i=1}^{k} \frac{1}{u_i} + \frac{M}{(M-1)u_{k+1}} \right) \cdot \text{opt}_{\text{BP}} + (M-1).$$

4.2 Reduction

In this subsection, we prove the following Theorem 14.

▶ **Theorem 14.** *Assume that the bounded space online bin-packing problem admits a solution using at most $\beta_M \cdot \mathrm{opt}_{\mathrm{BP}} + \gamma_M$ bins, where β_M and γ_M are parameters depending on the space bound M, and $\mathrm{opt}_{\mathrm{BP}}$ is defined in Section 4.1. Let $\varepsilon > 0$. Then Algorithm 1 uses at most $(\beta_M + 30\varepsilon^2 \cdot M) \cdot \mathrm{opt} + \gamma_M$ tours for the tree DVRP.*

Consider an instance of the DVRP on a tree. Let m denote the number of components, and let $\mathcal{C} = \{c_1, c_2, ..., c_m\}$ denote the set of components. For each component $c_i \in \mathcal{C}$, let $\bar{\ell}_{i,1}, \bar{\ell}_{i,2}, ..., \bar{\ell}_{i,e_i}$ denote the reduced lengths of the ending subtours in c_i from OPT, and $\bar{\ell}'_{i,1}, \bar{\ell}'_{i,2}, ..., \bar{\ell}'_{i,p_i}$ denote the reduced lengths of the passing subtours in c_i from OPT. We define the following instance of the bounded space online bin-packing:

$$L = (\bar{\ell}_{1,1}, \bar{\ell}_{1,2}, ..., \bar{\ell}_{1,e_1}, \bar{\ell}'_{1,1}, \bar{\ell}'_{1,2}, ..., \bar{\ell}'_{1,p_1}, ..., \bar{\ell}_{m,1}, \bar{\ell}_{m,2}, ..., \bar{\ell}_{m,e_m}, \bar{\ell}'_{m,1}, \bar{\ell}'_{m,2}, ..., \bar{\ell}'_{m,p_m}).$$

Let B_1 denote a solution to this instance satisfying the assumption of the claim, i.e.,

$$|B_1| \leq \beta_M \cdot \mathrm{opt}_{\mathrm{BP}} + \gamma_M. \tag{4}$$

For each bin b in B_1, we partition its contents so that each part contains the reduced lengths of the subtours that come from the same component and are in the same category, and we replace b by a collection of bins, one for each part of the partition containing at least one subtour of b. This defines a bin-packing B_2.

Next, we define a solution S to the tree DVRP corresponding to B_2. For each bin b of B_2, we consider the subtours that correspond to the reduced lengths in b, and we create one tour in S, which is the minimum tour visiting all terminals covered by those subtours. Observe that those subtours are in the same component and of the same category and, in addition, their reduced lengths sum to at most 1. By Lemma 10, the created tour is within the distance constraint D. Therefore, S is a feasible solution to the tree DVRP.

By construction, each tour in S visits terminals from only one component. Therefore, the output of Algorithm 1 is at most $|S|$, so is at most $|B_2|$. It remains to analyze $|B_2|$.

▶ **Lemma 15.** $|B_2| \leq |B_1| + (30/\Gamma)M \cdot \mathrm{opt}$.

Proof. Let $J \subseteq [1, |L| - 1]$ denote the set of integers $j \leq |L| - 1$ such that the j^{th} element in L and the $(j+1)^{\text{th}}$ element in L are the reduced lengths of two subtours in different components or of different categories. From the construction of L and since there are m components and two categories, we have

$$|J| \leq 2m. \tag{5}$$

Consider a bin $b \in B_1$. The number of bins in B_2 generated by b is the number of pairs (c, x) where $c \in \mathcal{C}$ and $x \in \{\text{passing, ending}\}$, such that b contains a reduced length of a subtour in component c and of category x. Let $\min(b)$ (resp. $\max(b)$) denote the minimum (resp. maximum) integer $j \in [1, |L|]$ such that the j^{th} element in L belongs to b. Let p_b denote the number of elements $j \in J$ such that $j \in [\min(b), \max(b)]$. The number of bins in B_2 generated by b is at most $1 + p_b$. Summing over all bins of B_1 gives

$$|B_2| \leq \sum_{b \in B_1} (1 + p_b) = |B_1| + \sum_{b \in B_1} p_b. \tag{6}$$

Observe that

$$\sum_{b \in B_1} p_b \leq \sum_{j \in J} \#\big(\text{bins } b \in B_1 \text{ such that } j \in [\min(b), \max(b)]\big).$$

For each $j \in J$, if a bin $b \in B_1$ is such that $j \in [\min(b), \max(b)]$, then b is *open* at the time of j. Since B_1 is a solution to the bounded space online bin packing, the number of open bins at the time of j is at most M, thus the number of bins $b \in B_1$ such that $j \in [\min(b), \max(b)]$ is at most M. Therefore,

$$\sum_{b \in B_1} p_b \leq |J| \cdot M \leq 2m \cdot M \leq (30/\Gamma)M \cdot \text{opt}, \tag{7}$$

where the second inequality follows from Equation (5) and the last inequality follows from Lemma 11.

The claim follows from Equations (6) and (7). ◀

Combining Lemma 15, Equation (4) and using $\Gamma = 1/\varepsilon^2$, we have

$$|B_2| \leq \beta_M \cdot \text{opt}_{\text{BP}} + \gamma_M + 30\varepsilon^2 M \cdot \text{opt}.$$

Next, we show that opt_{BP} is at most opt. For each tour t in an optimal solution OPT to the tree DVRP, if t visits components $c_1, ..., c_k$ with subtours' reduced lengths $\bar{\ell}_1, \bar{\ell}_2, ..., \bar{\ell}_k$, then using Lemma 9 we have $\bar{\ell}_1 + \bar{\ell}_2 + ... + \bar{\ell}_k \leq 1$. So it is possible to put the reduced lengths $\bar{\ell}_1, \bar{\ell}_2, ..., \bar{\ell}_k$ in a single bin of size 1. This leads to a feasible solution to the bin packing in the offline setting that uses opt bins. Thus $\text{opt}_{\text{BP}} \leq \text{opt}$. Hence

$$|B_2| \leq (\beta_M + 30\varepsilon^2 M) \cdot \text{opt} + \gamma_M.$$

This completes the proof of Theorem 14.

4.3 Proof of Theorem 2 Using the Reduction (Theorem 14)

First, consider the case when $\text{opt} < 1/\varepsilon^2$. Since $\Gamma = 1/\varepsilon^2$, Algorithm 2 returns a single component, let it be c. Then Algorithm 3 computes an optimal solution in c, thus the solution returned by Algorithm 1 is optimal.

Next, consider the case when $\text{opt} \geq 1/\varepsilon^2$. Let $M = 1/\varepsilon$. Let k be defined in Theorem 13. By Theorem 13, there exists a solution to the bounded space online bin-packing problem using at most $\beta_M \cdot \text{opt}_{\text{BP}} + \gamma_M$ tours, where

$$\beta_M := \left(\sum_{i=1}^{k} \frac{1}{u_i} + \frac{M}{(M-1)u_{k+1}} \right) \leq \alpha + 2\varepsilon, \text{ since } u_{k+1} \geq M,$$

$$\gamma_M := M - 1 < 1/\varepsilon.$$

Thus by the reduction (Theorem 14), the number of tours used by Algorithm 1 is at most

$$(\beta_M + 30\varepsilon^2 \cdot M) \cdot \text{opt} + \gamma_M < (\alpha + 2\varepsilon + 30\varepsilon) \cdot \text{opt} + 1/\varepsilon \leq (\alpha + 33\varepsilon) \cdot \text{opt},$$

where the last inequality follows since $\text{opt} \geq 1/\varepsilon^2$.

By Lemmas 5 and 7, the running time of Algorithm 1 is $O\left(n^2 \cdot \left(D/\varepsilon^2\right)^{O(1/\varepsilon^2)}\right)$.

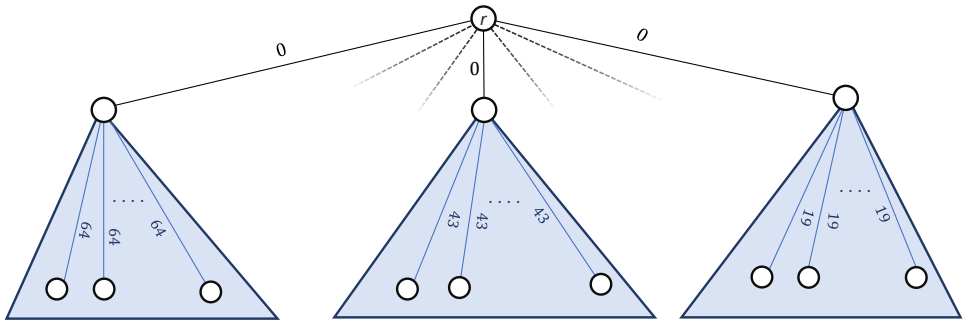


Figure 3 Instance $\mathcal{I}_k$ for $k = 3$. From Definition 1, $u_1 = 1$, $u_2 = 2$, $u_3 = 6$, $u_4 = 42$. The distance constraint $D = 2 \cdot k \cdot u_{k+1} = 252$. In a type-1 component, the edge weight $x_1 = 64$. In a type-2 component, the edge weight $x_2 = 43$. In a type-3 component, the edge weight $x_3 = 19$. Consider a tour t that visits a terminal in a type-1 component, a terminal in a type-2 component, and a terminal in a type-3 component. The length of t is $2(x_1 + x_2 + x_3) = D$.

5 Proof of Theorem 3

For each positive integer k, we construct an instance $\mathcal{I}_k$ of the tree DVRP as follows.

The tree in the instance $\mathcal{I}_k$ consists of a root vertex r and a set of components. The root of each component is connected to r by an edge of weight 0. The components are of k *types*. For each $i \in [1, k]$, a type-i component consists of $1 + \Gamma \cdot u_i$ vertices: One vertex is the root of the component, and the remaining $\Gamma \cdot u_i$ vertices are the terminals, each connected to the root of the component by an edge of weight $x_i := k \cdot u_{k+1}/(u_i + 1) + 1$. There are u_k/u_i components of type i for each $i \in [1, k]$. See Figure 3. Observe that $u_{k+1}/(u_i + 1)$ and u_k/u_i are both integers according to Definition 1. We set the distance constraint $D := 2k \cdot u_{k+1}$.

We claim that there exists a feasible solution to $\mathcal{I}_k$ using $\Gamma \cdot u_k$ tours. Consider a set of tours S such that each tour $t \in S$ visits exactly k terminals: for each $i \in [1, k]$, tour t visits exactly one terminal from all components of type i. For any $i \in [1, k]$, there are $\Gamma \cdot u_i \cdot u_k/u_i = \Gamma \cdot u_k$ terminals in all components of type i. Thus S consists of $\Gamma \cdot u_k$ tours. Next, we show that each tour $t \in S$ is within the distance constraint D. We have

$$\text{length}(t) = \sum_{i=1}^{k} 2x_i = 2k \cdot u_{k+1} \sum_{i=1}^{k} \frac{1}{u_i + 1} + 2k. \tag{8}$$

Using Definition 1, it is easy to show the following fact by induction.

▶ **Fact 16.** *For any positive integer k, we have*

$$\frac{1}{u_{k+1}} = 1 - \sum_{i=1}^{k} \frac{1}{u_i + 1}.$$

From Equation (8) and Fact 16, we have

$$\text{length}(t) = 2k \cdot u_{k+1} \left(1 - \frac{1}{u_{k+1}}\right) + 2k = D.$$

Therefore, S is a feasible solution. Hence

$$\text{opt} \leq \Gamma \cdot u_k. \tag{9}$$

Next, we analyze the solution to $\mathcal{I}_k$ computed by Algorithm 1. Recall that Algorithm 1 computes an optimal solution in each component independently. Let c be any component. Let $i \in [1, k]$ be the type of c. We observe that a tour is able to cover u_i terminals in c but is unable to cover $u_i + 1$ terminals in c. This is because, the cost to cover u_i terminals in c is

$$u_i \cdot 2x_i = D - \frac{2k \cdot u_{k+1}}{u_i + 1} + 2u_i \leq D,$$

where the inequality follows from Definition 1, and the cost to cover $u_i + 1$ terminals in c is

$$(u_i + 1) \cdot 2x_i = D + 2(u_i + 1) > D.$$

Since there are $\Gamma \cdot u_i$ terminals in c, the minimum number of tours to cover the terminals in c is Γ.

For each $i \in [1, k]$, the number of components of type i is u_k/u_i. Thus the number of tours returned by Algorithm 1 is $\sum_{i=1}^{k}(u_k/u_i) \cdot \Gamma = \Gamma \cdot u_k \sum_{i=1}^{k} 1/u_i$.

Combined with Equation (9), we conclude that the approximation ratio of Algorithm 1 on $\mathcal{I}_k$ is at least $\sum_{i=1}^{k} 1/u_i$, which tends to α when k tends to ∞ by Definition 1.

This completes the proof of Theorem 3.

References

1. Arjang A. Assad. Modeling and implementation issues in vehicle routing. In *Vehicle routing: Methods and studies*, pages 7–45, 1988.
2. Sean Barker, Timothy Wood, Prashant Shenoy, and Ramesh Sitaraman. An empirical study of memory sharing in virtual machines. In *USENIX Annual Technical Conference*, pages 273–284, 2012.
3. 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.
4. Teodor G. Crainic and Gilbert Laporte. *Fleet management and logistics*. Springer Science & Business Media, 2012.
5. Zachary Friggstad and Chaitanya Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. In *Proceedings of the forty-sixth annual ACM Symposium on Theory of Computing (STOC)*, pages 744–753, 2014.
6. Michael R. Garey and David S. Johnson. *Computers and Intractability: a guide to the theory of NP-Completeness*. Freeman, New York, 1985.
7. 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.
8. 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.
9. Gilbert Laporte, Martin Desrochers, and Yves Nobert. Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172, 1984.
10. Chan C. Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.
11. Chung-Lun Li, David Simchi-Levi, and Martin Desrochers. On the distance constrained vehicle routing problem. *Operations Research*, 40(4):790–799, 1992.

12 Claire Mathieu and Hang Zhou. A PTAS for capacitated vehicle routing on trees. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 95:1–95:20, 2022.

13 Viswanath Nagarajan and R Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 59(2):209–214, 2012.

14 Safraz Rampersaud and Daniel Grosu. Sharing-aware online virtual machine packing in heterogeneous resource clouds. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):2046–2059, 2016.

15 Michael Sindelar, Ramesh K. Sitaraman, and Prashant Shenoy. Sharing-aware algorithms for virtual machine colocation. In *Proceedings of the twenty-third annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 367–378, 2011.

16 Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. SIAM, 2002.

A NP-Hardness

▶ **Lemma 17.** *The Tree DVRP is strongly NP-hard.*

Proof. We reduce the bin packing problem to the tree DVRP. Consider an instance of the bin packing problem with an integer bin capacity M and n items of sizes $a_1, \ldots, a_n$, where $a_i \in [0, M]$ for each $i \in [1, n]$. The bin packing problem looks for a partition of the n items into the minimum number of bins such that in each bin, the sum of the item sizes is at most M. We construct an instance of the tree DVRP as follows. There is a depot and n terminals. For each $i \in [1, n]$, there is an edge between the depot and the i^{th} terminal with weight a_i. Let $D = 2M$. It is easy to see that a solution to the bin packing instance is equivalent to a soution to the tree DVRP instance. Since the bin packing problem is strongly NP-hard [6], the tree DVRP is strongly NP-hard. ◀

B Component Decomposition Algorithm

For each vertex v of the tree, let $T(v)$ denote the subtree rooted at v. For any subgraph H of the tree, let U_H denote the set of terminals in U that belong to the subgraph H.

The algorithm is given in Algorithm 2.

Algorithm 2 Decomposition algorithm $\textsc{Decompose}^{(\Gamma)}$ parameterized by Γ (see Lemma 5).

Input A tree T rooted at r, a distance constraint D
Output A decomposition of T into components

{Leaf components} := $\{T(v) : v$ least deep vertex s.t. $\textsc{Solve}^{(\Gamma)}(T(v), D - 2 \cdot \text{dist}(r, v), U_{T(v)}) \leq \Gamma\}$
$T' \leftarrow$ subtree spanning $\{r\} \cup \{$roots of leaf components$\}$
for each maximal downward v_1-to-v_2 path in T' whose internal vertices have only one child in T' **do**
    Let v_1' be the child of v_1 on the v_1-to-v_2 path.
    while $\textsc{Solve}^{(\Gamma)}(T(v) \setminus T(v_2), D - 2 \cdot \text{dist}(r, v), U_{T(v) \setminus T(v_2)}) \leq \Gamma$ **do**
        $v \leftarrow$ least deep vertex on the v_1'-to-v_2 path such that
            $\textsc{Solve}^{(\Gamma)}(T(v) \setminus T(v_2), D - 2 \cdot \text{dist}(r, v), U_{T(v) \setminus T(v_2)}) \leq \Gamma$
        Define internal component $(T(v) \setminus T(v_2))$ with exit vertex v_2
        $v_2 \leftarrow v$
    Define internal component $(v_1, v_1') \cup (T(v_1') \setminus T(v_2))$ with exit vertex v_2

Running time

The number of calls on $\textsc{Solve}^{(\Gamma)}$ is $O(n)$. Each call takes time $O(n \cdot (2\Gamma)^{\Gamma} \cdot D^{3\Gamma})$ by Lemma 7. Thus the overall running time of Algorithm 2 is $O(n^2 \cdot (2\Gamma)^{\Gamma} \cdot D^{3\Gamma})$.

C Proof of Theorem 7

Let $\tilde{T} = (\tilde{V}, \tilde{E})$ be a tree with root $\tilde{r}$. Let $\tilde{n}$ denote the number of vertices in $\tilde{T}$. Let $\tilde{U} \subseteq \tilde{V}$ be the set of terminals. The goal is to cover the terminals in $\tilde{U}$ using a minimum number of tours of length at most $\tilde{D}$ each.

Let Γ be a positive integer. We design a dynamic program that computes an optimal solution of at most Γ tours if such a solution exists. See Algorithm 3.

Algorithm 3 Dynamic program $\textsc{Solve}^{(\Gamma)}$ parameterized by Γ (see Lemma 7).

Input A tree $\tilde{T}$ rooted at $\tilde{r}$, a set of terminals $\tilde{U}$, a distance constraint $\tilde{D}$
Output $\min\{|S| : S$ is a feasible set of tours and $|S| \leq \Gamma\}$

Preprocess the tree $\tilde{T}$ so that each leaf is a terminal and each non-leaf vertex of $\tilde{T}$ has exactly two children ▷ Section 2.1
for each configuration (v, A) **do**
    $valid(v, A) = false$
for each terminal v in $\tilde{T}$ **do** ▷ Case 1
    for each list A such that $\ell(A) \in [1, \Gamma]$ and every element in A equals 0 **do**
        $valid(v, A) = true$
    for each non-terminal v of $\tilde{T}$ in bottom-up order **do** ▷ Case 2
        Let v_1 and v_2 denote the two children of v
        for each lists A, A_1, A_2 such that (v_1, A_1) and (v_2, A_2) are valid configurations **do**
            if (v_1, A_1), (v_2, A_2), and (v, A) are *compatible* **then** ▷ Definition 19
                $valid(v, A) \leftarrow true$
return $\min\{\ell(A) : (\tilde{r}, A)$ is valid$\}$

To begin with, using the preprocessing step in Section 2.1, we transform the tree $\tilde{T}$ so that the terminals are the same as the leaves in $\tilde{T}$ and every non-leaf vertex in $\tilde{T}$ has exactly two children.

We compute values at *configurations* (Definition 18), which are solutions restricted to a subtree of $\tilde{T}$.

▶ **Definition 18** (configurations). *A* configuration (v, A) *is defined by a vertex* $v \in \tilde{T}$ *and a list* A *of* $\ell(A)$ *integers* $(s_1, s_2, \ldots, s_{\ell(A)})$ *such that*

- $\ell(A) \leq \Gamma$;
- *for each* $i \in [1, \ell(A)]$, s_i *is an integer in* $[0, \tilde{D}]$.

We say that a configuration (v, A) is *valid* if it is possible to cover the terminals in the subtree of $\tilde{T}$ rooted at v with a collection of $\ell(A)$ subtours, such that each subtour starts and ends at v and the i-th subtour has length s_i. Note that $\ell(A)$ is equal to the total number of subtours in the collection. Thus the objective is to find a valid configuration $(\tilde{r}, A)$ with $\ell(A)$ minimum.

Let v be any vertex in $\tilde{T}$. We decide whether the configuration (v, A) is valid according to one of the two cases.

Case 1: v is a leaf vertex in $\tilde{T}$

Then v is a terminal. The configuration (v, A) is *valid* if and only if $\ell(A) \in [1, \Gamma]$ and every element in the list A equals 0.

Case 2: v is a non-leaf vertex in $\tilde{T}$

Let v_1 and v_2 be the two children of v in $\tilde{T}$.

▶ **Definition 19** (compatibility). *Let w_1 (resp. w_2) denote the weight of the edge between v and v_1 (resp. v_2). We say that the 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 elements such that at most one element is from A_1 (resp. A_2), and a one-to-one correspondence between every part in $\mathcal{P}$ and every element in A such that:*

- *a part in $\mathcal{P}$ consisting of one element $s^{(1)} \in A_1$ corresponds to an element s in A if and only if $s^{(1)} + 2w_1 = s$;*
- *a part in $\mathcal{P}$ consisting of one element $s^{(2)} \in A_2$ corresponds to an element s in A if and only if $s^{(2)} + 2w_2 = s$;*
- *a part in $\mathcal{P}$ consisting of two elements $s^{(1)}$ and $s^{(2)}$ corresponds to an element s in A if and only if $s = s^{(1)} + 2w_1 + s^{(2)} + 2w_2$.*

The configuration (v, A) is *valid* if and only if there exist a valid configuration (v_1, A_1) and a valid configuration (v_2, A_2) that are compatible with (v, A).

Running time

For each vertex $v \in \tilde{T}$, since $\ell(A) \leq \Gamma$, the number of configurations (v, A) is at most $\tilde{n} \cdot \tilde{D}^{\Gamma}$. For fixed $(v_1, A_1), (v_2, A_2)$, and (v, A), to check compatibility, there are at most $(2\Gamma)^{\Gamma}$ partitions of $A_1 \cup A_2$ into parts. Thus the overall running time of Lemma 7 is $O(\tilde{n} \cdot (2\Gamma)^{\Gamma} \cdot \tilde{D}^{3\Gamma})$.