# A Framework for Vehicle Routing Approximation Schemes in Trees

Amariah Becker[1(✉)] and Alice Paul[2]

[1] Computer Science Department, Brown University, Providence, USA
amariah_becker@brown.edu
[2] Data Science Initiative, Brown University, Providence, USA
alice_paul@brown.edu

**Abstract.** We develop a general framework for designing polynomial-time approximation schemes (PTASs) for various vehicle routing problems in trees. In these problems, the goal is to optimally route a fleet of vehicles, originating at a depot, to serve a set of clients, subject to various constraints. For example, in Minimum Makespan Vehicle Routing, the number of vehicles is fixed, and the objective is to minimize the longest distance traveled by a single vehicle. Our main insight is that we can often greatly restrict the set of potential solutions without adding too much to the optimal solution cost. This simplification relies on partitioning the tree into clusters such that there exists a near-optimal solution in which every vehicle that visits a given cluster takes on one of a few forms. In particular, only a small number of vehicles serve clients in any given cluster. By using these coarser building blocks, a dynamic programming algorithm can find a near-optimal solution in polynomial time. We show that the framework is flexible enough to give PTASs for many problems, including Minimum Makespan Vehicle Routing, Distance-Constrained Vehicle Routing, Capacitated Vehicle Routing, and School Bus Routing, and can be extended to the multiple depot setting.

**Keywords:** Approximation algorithms · Vehicle routing · Rooted tree cover

## 1 Introduction

Vehicle routing problems address the fundamental problem of routing a fleet of vehicles from a common depot to visit a set of clients. These problems arise naturally in many real world settings, and are well-studied across computer science and operations research. We generalize a class of vehicle routing problems by introducing the notions of *vehicle load*, the problem-specific vehicle constraint (e.g. vehicle capacity, distance traveled by the vehicle, client regret, etc.), and

Research funded by NSF grant CCF-14-09520.


Z. Friggstad et al. (Eds.): WADS 2019, LNCS 11646, pp. 112–125, 2019.
https://doi.org/10.1007/978-3-030-24766-9_9

*fleet budget*, the problem-specific fleet constraint (e.g. number of vehicles, sum of distances traveled, etc.).

Most vehicle routing problems can then be framed as either Min-Max Vehicle Load: minimize the maximum vehicle load, given a bound $k$ on fleet budget (e.g. Minimum Makespan Vehicle Routing) or Minimum Fleet Budget: minimize the required fleet budget, given a bound $D$ on vehicle load (e.g. Distance-Constrained Vehicle Routing). In fact, these are two optimization perspectives of the same decision problem: does there exist a solution with maximum vehicle load $D$ and fleet budget $k$?

### 1.1 Main Contributions

We present a framework for designing polynomial time approximation schemes (PTASs) for Min-Max Vehicle Load and Minimum Fleet Budget in trees. Tree (and treelike) transportation networks occur in building and warehouse layouts, mining and logging industries, and along rivers and coastlines [11,12]. Our framework applies directly to Min-Max Vehicle Load problems and generates results of the following form.

**Theorem 1.** *For every $\epsilon > 0$, there is a polynomial-time algorithm that, given an instance of* Min-Max Vehicle Load *on a tree, finds a feasible solution whose maximum vehicle load is at most $1 + \epsilon$ times optimum.*

An immediate corollary of Theorem 1 is the following result for the associated Minimum Fleet Budget problem.

**Theorem 2.** *Given an instance of* Minimum Fleet Budget *on a tree, if there exists a solution with fleet budget $k$ and vehicle load $D$, then for any $\epsilon > 0$, there is a polynomial-time algorithm that finds a solution with fleet budget $k$ and vehicle load at most $(1 + \epsilon)D$.*

The input to the framework is a rooted tree $T = (V, E)$ with root $r \in V$ and edge lengths $\ell(u, v) \geq 0$ for all $(u, v) \in E$. Without loss of generality, the root $r$ represents the *depot* at which all vehicles start and the set of clients corresponds to the set of leaves in the tree (we can add zero cost edges to ensure that every client is a leaf and any subtree without a client can be safely removed from the instance). Since every edge must then be traversed by at least one vehicle, the problems are equivalent to corresponding tree-cover problems.

As stated, the framework can be customized to a wide range of problems. In Sect. 4, we illustrate in detail how to customize the framework to give a PTAS for the Minimum Makespan Vehicle Routing problem of finding $k$ *tours* each starting and ending at a depot $r$ that serve all clients in $T$ such that the *makespan*, the maximum length of any tour, is minimized. Here, vehicle load is the tour length, and fleet budget is the number of vehicles. A bicriteria PTAS for the associated Minimum Fleet Budget problem, Distance-Constrained Vehicle Routing, follows as a corollary.

Our framework can be applied to give similar results for other vehicle-routing variants, including Capacitated Vehicle Routing and School Bus Routing, and can also be generalized to the multiple-depot setting. We state these results in Sect. 5 and refer the reader to the full version of our paper for details. The breadth of the problems listed highlights the real flexibility and convenience of the presented framework.

At a high level, the framework partitions the tree into *clusters* such that there exists a near-optimal solution that within each cluster has a very simple form, effectively coarsening the solution space. Then, given this simplified structure, a dynamic program can be designed to find such a near-optimal solution.

The clusters are designed to be *small* enough so that simplifying vehicle routes at the cluster level does not increase the optimal load by too much, but also *large* enough that the (coarsened) solutions can be enumerated efficiently. To bound the error introduced by this simplification we design a load-reassignment tool that makes cluster coverage adjustments *globally* in the tree.

Finally, standard dynamic programming techniques can result in a large accumulation of rounding error. To limit the number of times that the load of any single route is rounded, we introduce a *route projection* technique that essentially pays in advance for load that the vehicle *anticipates* accumulating, allowing the dynamic program to round only once instead of many times for this projected load.

### 1.2 Related Work

For trees, Minimum Makespan Vehicle Routing is equivalent to Minimum Makespan Rooted Tree Cover: the minimum makespan for rooted tree cover is exactly half the minimum makespan for vehicle routing, since tours traverse edges twice. Minimum Makespan Rooted Tree Cover is NP-hard even on star instances but admits an FPTAS if the number, $k$, of subtrees is constant [15] and a PTAS for general $k$ [10]. For covering a *general graph* with rooted subtrees, [6] provides a 4-approximation; this bound was later improved to a 3-approximation by [13]. For tree metrics, an FPTAS is known for constant $k$ [16], and a $(2+\epsilon)$-approximation is known for general $k$ [13]. In this paper, we improve this to a PTAS. Although a recent paper [4] also claimed to present a PTAS, in the full version of our paper we show that their result is incorrect and cannot be salvaged using the authors' proposed techniques. Additionally, we compare their approach to our own and describe how we successfully overcome the challenges where their approach fell short.

The associated Distance-Constrained Vehicle Routing problem is to minimize the number of tours of length at most $D$ required to cover all client demand. Even restricted to star instances, this problem is NP-hard, and for tree instances it is hard to approximate to better than a factor of $3/2$ [14]. A 2-approximation is known for tree instances, and $O(\log D)$ and $O(\log |S|)$-approximations are known for general metrics, where $S$ is the set of clients [14]. Allowing a multiplicative stretch in the distance constraint, a $(O(\log 1/\epsilon), 1+\epsilon)$ bicriteria approximation is also known, which finds a solution of at most

$O(\log 1/\epsilon)OPT_D$ tours each of length at most $(1+\epsilon)D$ [14], where $OPT_D$ is the minimum number of tours of length at most $D$ required to cover all clients. We give a $(1, 1+\epsilon)$ bicriteria PTAS for trees, and note that the hardness results for trees described above [14] imply that without allowing this $(1+\epsilon)$ stretch in $D$, a PTAS is unlikely to exist.

In the classic Capacitated Vehicle Routing each vehicle can cover at most $Q$ clients, and the objective is to minimize the *sum* of tour lengths. This problem is also NP-hard, even in star instances [12]. For tree metrics, a 4/3-approximation is known [2], which improves upon the previous best-known approximation ratio of $(\sqrt{41}-1)/4$ by [1] and is tight with respect to the best known lower bound. In this paper, we give a $(1, 1+\epsilon)$ bicriteria PTAS for trees. For general metrics, a $(2.5 - \frac{1.5}{Q})$-approximation is known [9](using [5]).

The *regret* of a path is the difference between the path length and the distance between the path endpoints. The Min-Max Regret Routing problem is to cover all clients with $k$ *paths* starting from the depot, such that the maximum regret is minimized. For trees, there is a known 13.5-approximation algorithm [3], which we improve to a PTAS in this paper. For general graphs there is a $O(k^2)$-approximation algorithm [7].

In the related School Bus Routing problem, there is a bound $R$ on the regret of each path and the goal is to find the minimum number of paths required to cover all client demand. For general graphs, [8] provides an LP-based 15-approximation algorithm, improving upon the authors' previous 28.86-approximation algorithm [7]. In trees, there exists a 3-approximation algorithm for the uncapacitated version of this problem and a 4-approximation algorithm for the capacitated version [3]. Additionally, there is a (3/2) inapproximability bound [3]. A PTAS is therefore unlikely to exist for trees. Instead, we give a $(1, 1+\epsilon)$ bicriteria PTAS that allows a $(1+\epsilon)$ stretch in the regret constraint.

## 2 Preliminaries

Let $OPT$ denote the value of an optimum solution. For a minimization problem, a polynomial-time $\alpha$-approximation algorithm is an algorithm that finds a solution of value at most $\alpha \cdot OPT$ and runs in time that is polynomial in the size of the input. A polynomial-time approximation scheme (PTAS) is a family of $(1+\epsilon)$-approximation algorithms indexed by $\epsilon > 0$ such that for each $\epsilon$, the algorithm runs in time polynomial in the input size, but may depend arbitrarily on $\epsilon$.

In a rooted tree, the *parent* of a vertex $v$, denoted $p(v)$, is the vertex adjacent to $v$ in the shortest path from $v$ to $r$ (the parent of $r$ is undefined). If $u = p(v)$ then $v$ is a *child* of $u$. The parent edge of a vertex $v$ is the edge $(p(v), v)$ (undefined for $v = r$). The *ancestors* of vertex $v$ are all vertices (including $v$ and $r$) in the shortest $v$-to-$r$ path and the *descendants* of $v$ are all vertices $u$ such that $v$ is an ancestor of $u$. We assume every vertex has at most two children. If vertex $v$ has $l > 2$ children $v_1, ..., v_l$, add vertex $v'$ and edge $(v, v')$ of length zero and replace edges $(v, v_1)$, $(v, v_2)$ with edges $(v', v_1)$, $(v', v_2)$ of the same lengths.

Further, the *subtree rooted at* $v$ is the subgraph induced by the descendants of $v$ and is denoted $T_v$. If $u = p(v)$, the $v$-*branch* at $u$ consists of the subtree

rooted at $v$ together with the edge $(u, v)$. We define the *length* of a subgraph $A \subseteq E$ to be $\ell(A) = \sum_{(u,v)\in A} \ell(u, v)$. For vertices $u, v$, we use $d_T(u, v)$ to denote the shortest-path distance in $T$ between $u$ and $v$.

Our framework applies to vehicle routing problems that can be framed as Min-Max Vehicle Load problems, in which the objective is to minimize the maximum vehicle load, subject to a fleet budget. Given a Min-Max Vehicle Load problem, a trivial $n$-approximation can be used to obtain an upper bound $D_{high}$ for $OPT$. An overarching algorithm takes as input a load value $D \geq 0$ and provides the following guarantee: if there exists a solution with max load $D$, the algorithm will find a solution with max load at most $(1 + \epsilon)D$. A PTAS follows from using binary search between $\frac{D_{high}}{n}$ and $D_{high}$ for the smallest value $D_{low}$ such that the algorithm returns a solution of max load at most $(1+\epsilon)D_{low}$. This implies $D_{low} \leq OPT$. For the rest of the paper, we assume $D$ is fixed.

## 3 Framework Overview

Optimization problems on trees are often well suited for dynamic programming algorithms. In fact, the following dynamic programming strategy can solve Min-Max Vehicle Load problems on trees exactly: at each vertex $v$, for each value $0 \leq i \leq D$, *guess* the number of solution route segments of load exactly $i$ in the subtree rooted at $v$. Such an algorithm would be exponential in $D$. Instead of considering every possible load value, route segment loads can be *rounded* up to the nearest $\theta D$, for some value $\theta \in (0, 1]$ that depends only on $\epsilon$, so that only $O(\theta^{-1})$ segment load values need to be considered. In order to achieve a PTAS, we must show that this rounding does not incur too much error. Rounding the load of a route at *every* vertex accumulates too much error, but if the number of times that any given route is rounded is at most $\epsilon/\theta$, then at most $\epsilon D$ error accumulates, as desired.

One main insight underlying our algorithm is that a route only needs to incur rounding error when it branches. The challenge in bounding the rounding error then becomes bounding the number of times a route branches. While a route in the optimal solution may have an arbitrary amount of branching, we show that we can greatly limit the scope of candidate solutions to those with a specific structure while only incurring an $\epsilon D$ error in the maximum load. Rather than having to make decisions for covering every leaf in the tree (of which there may be arbitrarily many−each with arbitrarily small load), we partition the tree into *clusters* and then address covering the clusters.

By *reassigning* small portions of routes within a cluster, we show that there exists a near-optimal solution in which all clients (leaves) within a given cluster are covered by only one or two vehicles. These clusters are chosen to be small enough that the error incurred by the reassignment is small, but large enough that any given route covers clients in a bounded number of clusters. This *coarsens* the solutions considered by the algorithm, as vehicles must commit to covering larger fractions of load at a time. A dynamic program then finds the optimal such coarse solution using these simple building blocks within each cluster.

### 3.1 Simplifying the Solution Structure

Let $\hat{\epsilon}$ and $\delta$ be problem-specific values that depend only on $\epsilon$. Let $\mathcal{H}_{\mathcal{T}}$ denote the set of all subgraphs of $T$, and let $g : \mathcal{H} \to \mathbb{Z}^{\geq 0}$ be a problem-specific *load function*. We require $g$ to be monotonic and subadditive. Intuitively, for all $H \in \mathcal{H}_{\mathcal{T}}$, $g(H)$ is the load accumulated by a vehicle for covering $H$.

**Condensing the Input Tree.** The first step in the framework is to CONDENSE all small branches into leaf edges. Specifically, let $\mathcal{B}$ be the set of all maximal branches of load at most $\delta D$. That is, for every $v$-branch $b \in \mathcal{B}$, $g(b) \leq \delta D$ and for $b$'s parent $p(v)$-branch, $b_p$, $g(b_p) > \delta D$. For convenience, if $b_1 \in \mathcal{B}$ is a $v_1$ branch at $u$ and $b_2 \in \mathcal{B}$ is a *sibling* $v_2$ branch at $u$ such that $g(b_1) + g(b_2) \leq \delta D$, we add a vertex $u'$ and an edge $(u, u')$ of length zero and replace $(u, v_i)$ with edge $(u', v_i)$ of length $\ell(u, v_i)$ for $i \in \{1, 2\}$. The $u'$ branch at $u$ then replaces the two branches $b_1$ and $b_2$ in $\mathcal{B}$. This ensures that any two branches in $\mathcal{B}$ with the same parent cannot be combined into a subtree of load $\leq \delta D$.

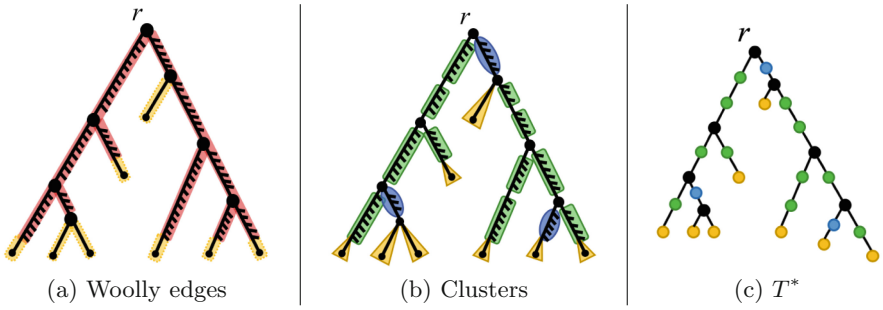
Then, for every $b \in \mathcal{B}$, we *condense* $b$ by replacing it with a leaf edge of length $\ell(b)$ and load $g(b)$. All clients in $b$ are now assumed to be co-located at the leaf. Though it is easier to think of these condensed branches as leaf edges, the algorithm need not actually modify the input tree; condensing a branch is equivalent to requiring a single vehicle to cover the entire branch.

**Clustering the Condensed Tree.** After condensing all small branches, we partition the condensed tree into clusters and define every leaf edge whose load is at least $\frac{\delta}{2}D$ to be a *leaf cluster*. The leaf-cluster-to-root paths define what we call the *backbone* of $T$. By construction, every edge that is not on this backbone is either a leaf cluster (of load $\geq \frac{\delta}{2}D$) or a leaf edge (of load $< \frac{\delta}{2}D$). That is, every vertex is at most one edge away from the backbone (see Fig. 1a).

We can think of the condensed tree as a binary tree whose root is the depot, whose leaves are the leaf clusters, and whose internal vertices are the branching points of the backbone. Each edge of this binary tree corresponds to a maximal path of the backbone between these vertices, together with the small leaf edges off of this path (see Fig. 1a). To avoid confusion with tree edges, we call these path and leaf subgraphs *woolly edges*. A *woolly subedge* of a woolly edge consists of a subpath of the backbone and all incident leaf edges.

A woolly edge $e$ whose load $g(e)$ is less than $\frac{\hat{\epsilon}\delta}{2}D$ is called a *small cluster*. The remaining woolly edges have load at least $\frac{\hat{\epsilon}\delta}{2}D$. We partition each such woolly edge into one or more woolly subedges, which we call *edge clusters*, each with load in $[\frac{\hat{\epsilon}\delta}{2}D, \frac{\delta}{2}D]$. Backbone edges do not contain clients and can be subdivided as needed to ensure enough granularity in the tree edge lengths so that such a partition is always possible (see Fig. 1b).

For convenience, we label the components of edge clusters. Let $\mathcal{C}$ be the set of edge clusters. For any edge cluster $C \in \mathcal{C}$, let $P_C$ denote the backbone path in $C$ and let $L_C$ denote the leaf edges in $C$. We order the backbone edges along $P_C$ as $p_{C,1}, p_{C,2}, ..., p_{C,m}$ in increasing distance from the depot and similarly label




(a) Woolly edges (b) Clusters (c) $T^*$

**Fig. 1.** (a) Leaf clusters in yellow and woolly edges in red; (b) The tree partitioned into leaf clusters (yellow triangles), small clusters (blue ovals), and edge clusters (green rectangles); (c) The corresponding $T^*$ for clustering from (b). (Color figure online)

the leaf edges $e_{C,1}, e_{C,2}, ..., e_{C,m-1}$ such that $e_{C,i}$ is the leaf incident to $p_{C,i}$ and $p_{C,i+1}$ for all $1 \leq i < m$ (see Fig. 2). If no such incident leaf exists for some $i$, we can add a leaf of length zero. Likewise $P_C$ can be *padded* with edges of length zero to ensure that each edge cluster 'starts' and 'ends' with a backbone edge.

**Solution Structure.** Consider the intersection of a solution with an edge cluster $C$. There are three different *types* of routes that visit $C$ (see Fig. 2). A $C$-*passing* route traverses $C$ without covering any clients, and thus includes all of $P_C$ but no leaf edges in $L_C$. A $C$-*collecting* route traverses *and* covers clients in $C$, and thus includes all of $P_C$ and some edges in $L_C$. Last, a $C$-*ending* route covers clients in, but does not traverse $C$, and thus includes backbone edges $p_{C,1}, p_{C,2}, ..., p_{C,i}$ for some $i < m$ and some leaves in $L_C$, but does not include all of $P_C$. Note that any $C$-*ending* route can be assumed to cover some leaves in $L_C$ because otherwise, removing any such redundancy would only improve a solution.

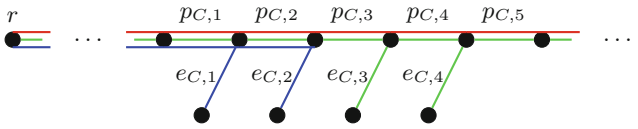



**Fig. 2.** Three types of route within an edge cluster $C$; the red tour is a $C$-passing route, the green tour is a $C$-collecting route, and the blue tour is a $C$-ending route. (Color figure online)

We say that a cluster $C$ has *single coverage* if a single vehicle covers *all* clients in $C$. We say that an edge cluster $C$ has *split coverage* if there is one $C$-*ending* route that covers leaf edges $e_{C,1}, e_{C,2}, ..., e_{C,i}$ for some $i < m-1$ and one $C$-*collecting* route that covers leaf edges $e_{C,i+1}, e_{C,i+2}, ..., e_{C,m-1}$ (see Fig. 2).

Finally, we say that a feasible solution has a *simple structure* if:

- Leaf clusters and small clusters have single coverage,
- Edge clusters have single or split coverage, and
- Each vehicle covers clients in $O(\frac{1}{\hat{\epsilon}^2\delta})$ clusters

Customization of the framework requires proving a *structure theorem* stating that there exists a near-optimal solution (i.e. a feasible solution with maximum load at most $(1+\epsilon)D$) with simple structure. Such a theorem proves that it is safe to reduce the set of potential solutions to those with simple structure.

### 3.2 Dynamic Program

After proving a structure theorem, the framework uses a dynamic programming algorithm (DP) to actually find a near-optimal solution with simple structure. We define the *cluster tree* $T^*$ to be the tree that results from contracting each cluster of $T$ to a single vertex. That is, the cluster tree has a vertex for each cluster and each branching point of the backbone (See Fig. 1c). The DP traverses $T^*$ starting at the leaves and moving rootward, and enumerates the possible route structures within each cluster. Namely, the DP considers all ways edge cluster coverage can be split and how routes are merged at branching points.

At each vertex in this tree the algorithm stores a set of *configurations*. A configuration is interpreted as a set of routes in $T$ that cover all clusters in the subtree of $T^*$ rooted at $v$. Let $\theta \in (0, 1]$ be a problem-specific value that depends only on $\epsilon$. A configuration at a vertex $v$ specifies, for each multiple $i$ of $\theta D$ between 0 and $(1+\epsilon)D$ the number of routes whose rounded load is $i$ at the time they reach $v$. Because $\theta$ depends only on $\epsilon$, the number of configurations and runtime of the DP is polynomially bounded. After traversing the entire cluster tree, the solution is found at the root. If there exists a configuration at the root such that all of the rounded route loads are at most $(1+\epsilon)D$, the algorithm returns this solution.

To ensure that the DP actually finds a near-optimal solution, we must bound the number of times that a given route is rounded to $\epsilon/\theta$, which gives a rounding error of at most $\epsilon D$. In particular, we design the DP so that the number of times that any one route is rounded is proportional to the number of clusters that it covers clients in. Then, using the structure theorem, there exists a near-optimal solution that covers clients in $O(\frac{1}{\hat{\epsilon}^2\delta})$ clusters and gets rounded by the DP $O(\frac{1}{\hat{\epsilon}^2\delta})$ many times. Finally, $\theta$ is set to $c_\theta \epsilon \hat{\epsilon}^2 \delta$ for some constant $c_\theta$.

For loads involving distance, $C$-passing routes pose a particular challenge for bounding rounding error. These routes may accumulate load while passing through clusters without covering any clients, yet the DP cannot afford to update the load at every such cluster. Instead, the DP *projects* routes to predetermined destinations up the tree, so that they accumulate rounding error only once while passing many clusters. The configuration then stores the (rounded) loads of the *projected* routes, and the DP need not update these load values for clusters passed through along the projection.

### 3.3 Reassignment Lemma

We now present a lemma that will serve as a general-purpose tool for our framework. This tool is used to *reassign* small route segments. That is, if some subgraph $H$ is covered by several small route segments from distinct vehicles $h_1, h_2, ..., h_m$, then for some $1 \leq i \leq m$, the entire subgraph $H$ is assigned to be covered by $h_i$. This *increases* load on $h_i$ so as to cover all of $H$, and *decreases* load on $h_j$ for all $j \neq i$ which are no longer required to cover $H$ (see Fig. 3). We show that this assignment process can be performed simultaneously for many such subgraphs such that the net load increase of any one route is small.

Let $G = (A, B, E)$ be an edge-weighted bipartite graph where $A$ is a set of *facilities*, $B$ is a set of *clients*, and $w(a, b) \geq 0$ is the weight of edge $(a, b) \in E$. For any vertex $v$, we use $N(v)$ to denote the *neighborhood* of $v$, namely the set of vertices $u$ such that there is an edge $(u, v) \in E$. Each facility $a \in A$ has capacity $q(a) = \sum_{b \in N(a)} w(a, b)$ and each client $b \in B$ has weight $w(b) \leq \sum_{a \in N(b)} w(a, b)$. A feasible *assignment* is a function $f : B \to A$, such that each client $b$ is assigned to an adjacent facility $f(b) \in N(b)$. We can think of the weights $w(a, b)$ representing fractional assignment costs while weight $w(b)$ corresponds to a "discounted" cost of wholly serving client $b$. Ideally, the total weight of clients assigned to any facility $a$ would not exceed the capacity $q(a)$; however, this is not always possible. We define the *overload* $h_f(a)$ of a facility $a$ to be $w(f^{-1}(a)) - q(a) = \sum_{b|f(b)=a} w(b) - \sum_{b \in N(a)} w(a, b)$ and the *overload* $h_f$ of an assignment to be $\max_{a \in A} h_f(a)$. The Bipartite Weight-Capacitated Assignment problem is to find an assignment with minimum overload.

**Lemma 1.** *Given an instance of the* Bipartite Weight-Capacitated Assignment *problem, an assignment with overload at most* $\max_{b \in B} w(b)$ *can be found efficiently.*

In our application of Lemma 1, facilities represent tours and clients represent subgraphs of $T$. Assignment of a client $b$ to a facility $a$ represents assigning subgraph $b$ to be covered by tour $a$ (see the proof of Lemma 2 for an example).

## 4 Customizing the Framework: Minimum Makespan Vehicle Routing

In this section, we demonstrate how to apply the general framework to a specific problem, Minimum Makespan Vehicle Routing. In particular, we use the framework to achieve the following:

**Theorem 3.** *For every* $\epsilon > 0$*, there is a polynomial-time algorithm that, given an instance of* Minimum Makespan Vehicle Routing *on a tree, finds a solution whose makespan is at most* $1 + \epsilon$ *times optimum.*

Recall that the problem is to find $k$ *tours* that serve all clients in $T$ such that the maximum *length* of any tour is minimized. The vehicle routes are tours,

and the vehicle load is tour length, so the load $g(H)$ of subgraph $H$ is twice the length of edges in the subgraph. The CONDENSE operation is then applied to the input tree, with $\delta = \hat{\epsilon} = \epsilon/c$ for some constant $c$ we will define later. Leaf clusters therefore correspond to branches of length at least $\frac{\hat{\epsilon}}{4}D$ (load at least $\frac{\hat{\epsilon}}{2}D$), small clusters have total length less than $\frac{\hat{\epsilon}^2}{4}D$, and edge clusters have total length in $[\frac{\hat{\epsilon}^2}{4}D, \frac{\hat{\epsilon}}{4}D]$. As described in Sect. 3, the two steps in applying the framework are proving a structure theorem and designing a dynamic program.

### 4.1 Minimum Makespan Vehicle Routing Structure Theorem

We prove the following for Minimum Makespan Vehicle Routing.

**Theorem 4.** *If there exists a solution with makespan $D$, then there exists a solution with makespan at most $1 + O(\hat{\epsilon})D$ that has simple structure.*

We prove the above by starting with some optimal solution of makespan at most $D$ and show that after a series of steps that transforms the solution into one with simple structure, the makespan is still near-optimal.

To ensure that each step maintains solution feasibility, we introduce the following notion of independence. Let $T'$ be a connected subgraph of $T$ containing the depot $r$, and let $X$ be a set of subgraphs of $T$. We say that $X$ is a *tour-independent* set with respect to $T'$ if $T' \cup X'$ is connected for all $X' \subseteq X$. In particular, if $T'$ is the subgraph covered by a single tour then adding any subgraphs in $X'$ creates a new feasible tour.

**Lemma 2.** *The* CONDENSE *operation adds at most $\hat{\epsilon}D$ to the optimal makespan.*

*Proof.* The CONDENSE operation is equivalent to requiring every branch in $\mathcal{B}$ to be covered by a single tour. We show that there is such a solution of makespan at most $OPT + \hat{\epsilon}D$. Fix an optimal solution, and let $A$ be the set of tours in the optimal solution that (at least partially) cover branches in $\mathcal{B}$. We define an edge-weighted bipartite graph $G = (A, \mathcal{B}, E)$ where there is an edge $(a, b)$ if and only if tour $a$ contains edges of branch $b$, and $w(a, b)$ is the length of the tour segment of $a$ in branch $b$, namely twice the length of the edges covered by tour $a$. Note that $\forall a \in A, b \in \mathcal{B}$, $w(a, b) \leq \hat{\epsilon}D$. For each $b \in \mathcal{B}$, we define the weight $w(b)$ to be $2\ell(b)$, and for each $a \in A$, we define the capacity $q(a)$ to be the sum $\sum_{b:a\cap b\neq\emptyset} w(a, b)$ of all tour segments of $a$ in branches of $\mathcal{B}$. Clearly, $w(b) \leq \sum_{a:a\cap b\neq\emptyset} w(a, b)$, since these tour segments collectively cover $b$.

Essentially, $q(a)$ represents tour $a$'s *budget* for buying whole branches and is defined by the length of its tour segments in the branches that it partially covers. Further, we will only assign a branch to a tour that already covers some edges in the branch so there is no additional cost to connect the tour to the branch.

Applying Lemma 1 to $G$, we can achieve an assignment of branches to tours such that each branch is assigned to one tour and the capacity of each tour is exceeded by at most $\max_{b\in B} w(b) \leq \hat{\epsilon}D$. Further, for any tour $a \in A$, let $T'_a$ be the corresponding subgraph visited by $a$ excluding any branches in $B$.

$T'_a$ contains $r$ and is connected, so $N_G(a) \subseteq \mathcal{B}$ is a tour-independent set with respect to $T'_a$. Thus, the reassignment of branches creates a feasible solution in which the extra distance traveled by each tour is at most $\hat{\epsilon}D$.

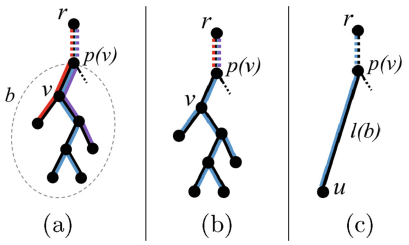



**Fig. 3.** (a) depicts a branch $b \in \mathcal{B}$ covered by several small tour segments; (b) shows the entire branch $b$ being assigned to the blue tour; (c) shows the result of the condense operation. (Color figure online)

**Lemma 3.** *Requiring all leaf clusters and small clusters to have single coverage increases the makespan by at most* $4\hat{\epsilon}D$.

*Proof.* After condensing the tree, all leaf clusters have single coverage, and the effect on makespan was covered in Lemma 2. Because of the binary tree structure, we can *assign* each small cluster to a descendant leaf cluster in such a way that each leaf cluster is assigned at most two small clusters. Since each leaf cluster is covered by a single tour, we can require this tour to also cover the clients of the small cluster(s) assigned to that leaf cluster. This is feasible since small clusters are only assigned to *descendant* leaf clusters. Furthermore, since leaf clusters have length at least $\frac{\hat{\epsilon}}{4}D$, we can *charge* this error to the length of the leaf clusters. In particular, since any given tour covers at most $D/(2 \cdot \frac{\hat{\epsilon}}{4}D) = \frac{2}{\hat{\epsilon}}$ leaf clusters, this assignment adds at most $2 \cdot \frac{2}{\hat{\epsilon}} \cdot (2 \cdot \frac{\hat{\epsilon}^2}{2}D) = 4\hat{\epsilon}D$ to the makespan.

**Lemma 4.** *Requiring every edge cluster to have single or split coverage adds at most* $3\hat{\epsilon}D$ *to the optimal makespan.*

After proving Lemma 4, all that remains in proving Theorem 4 is to bound the number of clusters that a single vehicle covers clients in. See the full version of our paper for proofs.

### 4.2 Minimum Makespan Vehicle Routing Dynamic Program

Having proven a structure theorem, we now present a dynamic programming algorithm (DP) that actually finds a near-optimal solution with simple structure.

Recall, the DP traverses cluster tree $T^*$ starting at the leaves and moving rootward. A configuration is a vector in $\{0, 1, 2, ..., k\}^{2\hat{\epsilon}^{-4}}$. A configuration $\boldsymbol{x}$ at a vertex $v$ is interpreted as a set of tours *projected up to* $r$ in $T$ that cover all clusters in the subtree of $T^*$ rooted at $v$. For $i \in \{1, 2, ..., 2\hat{\epsilon}^{-4}\}$, $\boldsymbol{x}[i]$ is the

number of tours in the set that have *rounded* length $i\hat{\epsilon}^4 D$. That is, the actual tours that correspond to the $\boldsymbol{x}[i]$ tours represented in the vector each have length that may be less than $i\hat{\epsilon}^4 D$.

The algorithm categorizes the vertices into three different cases and handles them separately. The *base cases* are the leaves of $T^*$. Let $v \in T^*$ be such a leaf, let $L_v$ be the corresponding leaf cluster in $T$, and let $u$ be the vertex at which $L_v$ meets the backbone. When the algorithm determines the configuration for $v$ it addresses covering both $L_v$ as well as covering any small clusters $C_1, ..., C_h$ that are assigned to $L_v$. Let $\ell_{small}$ be the length of all of the *leaves* of these small clusters, namely $\ell_{small} = \ell(\bigcup_{1 \leq i \leq h} C_i \setminus \text{backbone})$. Let $\ell_0$ be $2(\ell(L_v) + \ell_{small} + d_T(u, r))$ rounded up to the nearest $\hat{\epsilon}^4 D$. The only configuration stored at $v$ is $\boldsymbol{x}$ such that $\boldsymbol{x}[\ell_0] = 1$ and $\boldsymbol{x}[j] = 0, \forall j \neq \ell_0$. All cluster lengths and distances to the depot can be precomputed in linear time, after which each base case can be computed in constant time.

The *grow cases* are the vertices in $T^*$ that correspond to edge clusters in $T$. Let $v \in T^*$ be such a vertex, and let $C_v$ be the corresponding edge cluster in $T$. Let $u$ be the root-most vertex in $C_v$, and let $v' \in T^*$ be the lone child vertex of $v$. Note that $v'$ may correspond to a branching backbone vertex, a leaf cluster or another edge cluster, but by construction, $v$ has exactly one child. Since $C_v$ has single or split coverage, at most two tours in any configuration at $v$ are involved in covering the leaves of $C_v$: all other tours in the configuration are $C_v$-passing tours, and their representation in the configuration remains unchanged. The algorithm considers all possible rounded tour lengths $\ell_1$ for a $C_v$-ending tour $t_1$ for the configuration (including not having such a tour) and for each such $t_1$, the algorithm considers all possible (rounded) lengths $\ell_2$ for an incoming $C_v$-collecting tour $t_2$, *before* the remaining length from covering leaves in $C_v$ is added to the tour. Given $\ell_1$ and $\ell_2$, the algorithm can easily compute the resulting rounded length $\ell_3$ of $t_2$ *after* covering its share of $C_v$ leaves. For each configuration $\boldsymbol{x}'$ for child vertex $v'$, the algorithm determines configuration $\boldsymbol{x}$ for $v$ such that $\boldsymbol{x}[\ell_1] = \boldsymbol{x}'[\ell_1] + 1$, $\boldsymbol{x}[\ell_2] = \boldsymbol{x}'[\ell_2] - 1$, $\boldsymbol{x}[\ell_3] = \boldsymbol{x}'[\ell_3] + 1$, and $\boldsymbol{x}[i] = \boldsymbol{x}'[i]$ otherwise. If the resulting $\boldsymbol{x}$ is feasible, it is stored at $v$. Since there are at most $2\hat{\epsilon}^{-4}$ options for $\ell_1$ and $\ell_2$ and at most $k^{2\hat{\epsilon}^{-4}}$ configurations at $v'$, the runtime for each grow case is $k^{O(\hat{\epsilon}^{-4})}$.

Finally, the *merge cases* are the vertices in $T^*$ that correspond to branching backbone vertices in $T$ as well as the depot. Let $v \in T^*$ be such a vertex, and let $u$ be the corresponding vertex in $T$. Let $v_1, v_2 \in T^*$ be the two children of $v$ in $T^*$. Every tour $t$ in a configuration at $v$ will either be a directly inherited tour $t_i$ of rounded length $\ell_i$ from a configuration at $v_i$ for $i \in \{1, 2\}$, or will be a *merging* of some tour $t_1$ from $v_1$ and some $t_2$ from $v_2$ with resulting length $\ell_1 + \ell_2 - 2\ell(u, r)$ rounded up to the nearest $\hat{\epsilon}^4 D$ (recall that $t_1$ and $t_2$ are tours from the depot so the subtracted amount addresses over-counting the path to the depot). For every possible $(\ell_1, \ell_2)$ (including lengths of zero to account for tours inherited by children), the algorithm considers how many tours at $v$ could have resulted from merging a tour of length $\ell_1$ from $v_1$ with a tour of length $\ell_2$ from $v_2$. Each of these possibilities corresponds to a configuration $\boldsymbol{x}_i$ at $v_i$ for $i \in \{1, 2\}$ and

to a merged configuration $\boldsymbol{x}$ at $v$. If $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are valid configurations stored at $v_1$ and $v_2$, respectively, then the algorithm stores $\boldsymbol{x}$ at $v$. There are $k^{4\hat{\epsilon}^{-8}}$ such possibilities, so the runtime of each merge case is $k^{O(\hat{\epsilon}^{-8})}$. Note that the dynamic program only considers storing feasible configurations $\boldsymbol{x}$ at vertex $v$ so the algorithm maintains that there are at most $k$ tours total.

Since for any $\epsilon > 0$ the DP has a polynomial runtime, the following lemma, which we prove in the full version of our paper, completes the proof of Theorem 3.

**Lemma 5.** *The dynamic program described above finds a tour with maximum makespan at most* $(1 + \epsilon)D$.

### 4.3 Distance-Constrained Vehicle Routing

Recall that the Distance-Constrained Vehicle Routing problem is to minimize the number of tours of length at most $D$ required to cover all clients. Since it is the Minimum Fleet Budget problem associated with Minimum Makespan Vehicle Routing, the following bicriteria PTAS follows as a corollary to Theorem 3.

**Theorem 5.** *Given an instance of* Distance-Constrained Vehicle Routing *on a tree, if there exists a solution with $k$ tours of length at most $D$, then for any $\epsilon > 0$, there is a polynomial-time algorithm that finds a solution with $k$ tours of length at most $(1 + \epsilon)D$.*

## 5 Framework Applications

In this section we give theorem statements for several other problems and extensions that can be solved using our framework. See the full version of our paper for details and proofs.

**Theorem 6.** *Given an instance of* Capacitated Vehicle Routing *on a tree, if there exists a solution of total length $k$ and capacity $Q$, then for any $\epsilon > 0$, there is a polynomial-time algorithm that finds a solution of total length $k$ and capacity at most $(1 + \epsilon)Q$.*

**Theorem 7.** *Given an instance of the* School Bus Routing *problem on a tree, if there exists a solution consisting of $k$ paths of regret at most $R$, then for any $\epsilon > 0$, there is polynomial-time algorithm that finds a solution consisting of $k$ paths of regret at most $(1 + \epsilon)R$.*

**Theorem 8.** *There is a polynomial-time 2-approximation for the* School Bus Routing *problem in trees.*

**Theorem 9.** *For every $\epsilon > 0$ and $\rho > 0$, there is a polynomial-time algorithm that, given an instance of $\rho$-*Depot Minimum Makespan Vehicle Routing *on a tree, finds a solution whose makespan is at most $1 + \epsilon$ times optimum.*

## References


1. Asano, T., Katoh, N., Kawashima, K.: A new approximation algorithm for the capacitated vehicle routing problem on a tree. J. Comb. Optim. **5**(2), 213–231 (2001). https://doi.org/10.1023/A:1011461300596
2. Becker, A.: A tight 4/3 approximation for capacitated vehicle routing in trees. CoRR abs/1804.08791 (2018). http://arxiv.org/abs/1804.08791
3. Bock, A., Grant, E., Könemann, J., Sanità, L.: The school bus problem on trees. Algorithmica **67**(1), 49–64 (2013). https://doi.org/10.1007/s00453-012-9711-x
4. Chen, L., Marx, D.: Covering a tree with rooted subtrees-parameterized and approximation algorithms. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 2801–2820. SIAM (2018)
5. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University Pittsburgh, PA, Management Sciences Research Group (1976)
6. Even, G., Garg, N., Könemann, J., Ravi, R., Sinha, A.: Min-max tree covers of graphs. Oper. Res. Lett. **32**(4), 309–315 (2004). https://doi.org/10.1016/j.orl.2003.11.010
7. Friggstad, Z., Swamy, C.: 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 2014, pp. 744–753. ACM, New York (2014). https://doi.org/10.1145/2591796.2591840, http://doi.acm.org/10.1145/2591796.2591840
8. Friggstad, Z., Swamy, C.: Compact, provably-good LPs for orienteering and regret-bounded vehicle routing. In: Eisenbrand, F., Koenemann, J. (eds.) IPCO 2017. LNCS, vol. 10328, pp. 199–211. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59250-3_17
9. Haimovich, M., Rinnooy Kan, A.: Bounds and heuristics for capacitated routing problems. Math. Oper. Res. **10**(4), 527–542 (1985)
10. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM (JACM) **34**(1), 144–162 (1987)
11. Karuno, Y., Nagamochi, H., Ibaraki, T.: Vehicle scheduling on a tree with release and handling times. Ann. Oper. Res. **69**, 193–207 (1997)
12. Labbé, M., Laporte, G., Mercure, H.: Capacitated vehicle routing on trees. Oper. Res. **39**(4), 616–622 (1991)
13. Nagamochi, H., Okada, K.: Approximating the minmax rooted-tree cover in a tree. Inf. Process. Lett. **104**(5), 173–178 (2007)
14. Nagarajan, V., Ravi, R.: Approximation algorithms for distance constrained vehicle routing problems. Networks **59**(2), 209–214 (2012)
15. Sahni, S.K.: Algorithms for scheduling independent tasks. J. ACM (JACM) **23**(1), 116–127 (1976)
16. Xu, L., Xu, Z., Xu, D.: Exact and approximation algorithms for the min-max k-traveling salesmen problem on a tree. Eur. J. Oper. Res. **227**(2), 284–292 (2013)