A Tight 4/3 Approximation for Capacitated Vehicle Routing in Trees

Amariah Becker[1]
Brown University Department of Computer Science, Providence, RI, USA
amariah_becker@brown.edu

Abstract

Given a set of clients with demands, the Capacitated Vehicle Routing problem is to find a set of tours that collectively cover all client demand, such that the capacity of each vehicle is not exceeded and such that the sum of the tour lengths is minimized. In this paper, we provide a 4/3-approximation algorithm for Capacitated Vehicle Routing on trees, improving over the previous best-known approximation ratio of $(\sqrt{41}-1)/4$ by Asano et al.[2], while using the same lower bound. Asano et al. show that there exist instances whose optimal cost is 4/3 times this lower bound. Notably, our 4/3 approximation ratio is therefore tight for this lower bound, achieving the best-possible performance.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Approximation algorithms, Graph algorithms, Capacitated vehicle routing

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2018.3

1 Intro

Vehicle-routing problems address how a service can best be provided to meet the demand from a set of clients. These problems arise very naturally in both commercial and public planning. Real world vehicle-routing problems must account for the capacities of the vehicles, which limit the amount of client demand that can be met in a single trip. We formalize this problem as finding tours in a graph:

Given a graph $G = (V, E)$, a specified *depot* vertex r, a *client set* S, an edge length function $l : E \to \mathbb{Z}^{\geq 0}$, a demand function $d : S \to \mathbb{Z}^{\geq 0}$ and $Q > 0$, the Capacitated Vehicle Routing problem is to find a set of tours of minimum total length such that each tour includes r and the tours collectively *cover* all demand at every client and such that no tour covers more than Q demand. A tour can only cover demand from clients along the tour, but it may pass by some clients without covering their demand.

There are two common variants of this problem: *splittable* and *unsplittable*. In the splittable variant, the demand of a client can be collectively covered by multiple tours, and in the unsplittable variant, the entire demand of a client must be covered by the same tour.

Both variants of this problem are NP-hard and therefore unlikely to admit a polynomial-time exact solution, but constant factor approximations can be found in polynomial time [9]. A natural question is whether better performance can be achieved for restricted graph classes. One line of research has focused on approximating Capacitated Vehicle Routing in trees. Though the problem remains NP-hard in trees [12], better constant-factor approximations have been found than for general metrics.

[1] Research funded by NSF grant CCF-14-09520

Hamaguchi and Katoh [10] noted a simple lower bound for the splittable-variant of Capacitated Vehicle Routing in trees: every edge must be traversed by at least enough vehicles to accommodate all demand from the clients that use the edge on the shortest path to the depot. They then use this lower bound (denoted LB) to give a 1.5 approximation [10]. Following this work, Asano et al. [2] use the same lower bound to achieve a $(\sqrt{41}-1)/4$ approximation. They also prove the following lemma (see Appendix C):

▶ **Lemma 1** ([2]). *There exist instances of* Capacitated Vehicle Routing *in trees whose optimal solution costs* $4/3 \cdot LB$.

This shows that the best possible approximation ratio using this lower bound would be a 4/3-approximation. Our result, stated in Theorem 2, achieves this ratio, and is therefore tight with respect to LB. No further improvements over our result can be made until a better lower bound is found.

▶ **Theorem 2.** *There is a polynomial-time 4/3 approximation for* Capacitated Vehicle Routing *in trees that is tight with respect to* LB.

1.1 Related Work

As Capacitated Vehicle Routing generalizes the Traveling Salesman Problem (TSP) (which is the special case of $Q = |V|$) it is NP-hard, and in general metrics is APX-hard [13].

For general metrics, a technique called *Iterated Tour Partitioning* starts with a TSP solution, partitions this tour into paths of bounded capacity, and then makes vehicle routing tours by adding paths from the depot to each endpoint of each path [9]. Iterated Tour Partitioning results in a polynomial time $(1 + (1 - 1/Q)\alpha)$-approximation for splittable Capacitated Vehicle Routing, where α is the approximation ratio of the TSP tour. A similar approach can be used for the unsplittable variant, resulting in a $(2 + (1 - 2/Q)\alpha)$-approximation [1]. Using Christofides' 1.5-approximation for TSP [6], these ratios are $(2.5 - \frac{1.5}{Q})$ and $(3.5 - \frac{3}{Q})$ respectively. No significant improvements over iterated tour partitioning are known for general metrics.

Even in trees, splittable Capacitated Vehicle Routing is NP-hard by a reduction from bin packing [12], and unsplittable Capacitated Vehicle Routing is NP-hard to even approximate to better than a 1.5-factor [8]. Since depth-first search trivially solves TSP optimally in trees, iterated tour partitioning already gives a $(2 - \frac{1}{Q})$-approximation for splittable demands in trees and a $(3 - \frac{2}{Q})$-approximation for unsplittable demands in trees. Labbe et al. improved this to a 2-approximation for the unsplittable-demand variant [12]. For splittable Capacitated Vehicle Routing in trees, Hamaguchi and Katoh [10] define a natural lower bound on the cost of the optimal solution and give a 1.5-approximation algorithm that yields a solution with cost at most 1.5 times this lower bound. Asano, Katoh, and Kawashima [2] improve the ratio to $(\sqrt{41}-1)/4$ using this same lower bound. They also show that there are instances whose optimal cost is, asymptotically, 4/3 times this lower bound value (see Appendix C for such an instance). This implies that if a 4/3-approximation algorithm exists that uses this lower bound it would be *tight* (i.e. best possible). In this paper we resolve the question as to whether or not such an an algorithm exists.

All of the above results allow for arbitrary capacity Q. Even for fixed capacity $Q \geq 3$, Capacitated Vehicle Routing is APX-hard in general metrics [3]. For fixed capacities, Capacitated Vehicle Routing is polynomial-time solvable in trees, but is NP-hard in other metrics. For instances in the Euclidean plane ($\mathbb{R}^2$), polynomial-time approximation

schemes (PTAS) are known for instances where Q is constant [9], Q is $O(\log n / \log\log n)$ [3], and Q is $\Omega(n)$ [3]. For higher-dimensional Euclidean spaces $\mathbb{R}^d$, a PTAS is known for when Q is $O(\log^{1/d} n)$ [11]. While a *quasi*-polynomial-time approximation (QPTAS) is known for arbitrary Q on instances in the Euclidean plane ($\mathbb{R}^2$) [7], no PTAS is known for arbitrary Q in *any* non-trivial metric.

Recently, the first approximation schemes for non-Euclidean metrics were designed. Specifically, a quasi-polynomial time approximation scheme is known for planar and bounded-genus graphs when Q is fixed [5], and a polynomial-time approximation scheme is known for graphs of bounded highway-dimension when Q is fixed [4].

1.2 Techniques

Our work extends the techniques introduced by Asano et al. [2] which itself was an extension of the work of Hamaguchi and Katoh [10]. Specifically, Hamaguchi and Katoh describe a very natural lower bound that arises on tree instances [10]: the number of tours that traverse each edge must be at least enough to cover all demand in the subtree below the edge (the tree is assumed to be rooted at the depot). This introduces a minimum *traffic* value on the edge. Multiplying this value by two (each tour crosses each edge once in each direction) times the weight of the edge and summing over all edges provides a lower bound on the cost of any feasible solution.

The algorithm of Asano et al. [2] proceeds in a sequence of rounds. In each round, a set of tours is identified such that the *cost-to-savings ratio*, namely the ratio of the cost (of these tours) to the reduction to the lower bound that results from taking these tours, is bounded by some constant α. The key is that although a given tour itself may cost more than α times its reduction to the lower bound, collectively the set of tours has the desired ratio. If after a round ends, no uncovered demand remains, then the union of these sets of tours is a feasible solution with cost at most α times the lower bound, which is at most α times the optimal cost. For Asano et al. [2], $\alpha = (\sqrt{41} - 1)/4$. We use a similar approach to achieve $\alpha = 4/3$.

Asano et al. [2] also introduced the idea of making *safe* modifications to the instance. That is, modifying the structure of the instance in such a way that does not increase the value of the lower bound or decrease the optimum cost (although it may increase the optimum cost) and such that a feasible solution in the modified instance has a corresponding feasible solution in the original instance. These modifications can be made safely at any point in the algorithm. Our algorithm also makes use of safe modifications, although the ones that we define differ somewhat from those defined by Asano et al. [2].

These modifications allow us to reason better about how the resulting instance must be structured. The idea is that the modifications can be made until one of a few cases arise. Each case has a corresponding *strategy* to find a set of tours with the desired cost-to-savings ratio.

Specifically, the algorithm of Asano et al. [2] classifies the *leafmost* subtrees containing at least $2Q$ units of demand into one of a few cases. The main obstacle in extending their algorithm to a 4/3-approximation is that one of the cases does not seem to have a good strategy. On the other hand, modifying the algorithm to instead classify the *leafmost* subtrees containing at least βQ units of demand for some $\beta > 2$ can greatly increase the number of cases that arise.

We overcome this obstacle by generalizing the difficult case into what we call a *p-chain* (See Figure 2). Our key insight is that even arbitrarily large p-chains can be addressed efficiently in sibling pairs and at the root. Our algorithm effectively delays addressing the difficult case by pushing it rootward until it finds a pair or reaches the root and is thus easy

to address. To keep the number of cases small, we address easy cases as they emerge, and require that any remaining difficult case must have a specific structure that the algorithm can easily detect in subsequent rounds.

2 Preliminaries

We use OPT to denote the cost of an optimization problem. For a minimization problem, a polynomial-time α-approximation algorithm is an algorithm with a runtime that is polynomial in the size of the input and returns a feasible solution with cost at most $\alpha \cdot OPT$.

When the input graph G is a tree it is assumed to be rooted at the depot r. Let $P[u, v]$ denote the unique path from u to v in the tree. Recall that the problem gives a length $l(e) \geq 0$ for each edge e, and let $l(P[u, v]) = \sum_{e \in P[u,v]} l(e)$ denote the shortest path distance between u and v.

For rooted tree $T = (V, E)$, and $v \in V$ let T_v denote the subtree rooted at v and $d(T_v)$ be the total demand from vertices in T_v.

The parent of a vertex v is the vertex u adjacent to v in $P[v, r]$, and the parent of r is undefined. An edge labeled (u, v) indicates that u is the parent of v. The parent of an edge (u, v) is the edge $(parent(u), u)$ and is undefined if $u = r$. If v has parent vertex u, we call $B = T_v \cup \{(u, v)\}$ a *branch* at u, and edge (u, v) the *stem* of the branch. The parent of a branch at u with stem e is the branch at $parent(u)$ with stem $parent(e)$ and is undefined if $u = r$. A vertex (resp. edge, branch) with a parent is said to be a child vertex (resp. edge, branch) of that parent.

2.1 Lower Bound

Here we describe the lower bound introduced by Hamaguchi and Katoh [10] and Asano et al. [2] and adopt similar terminology. Since all demand must be covered and each tour can cover at most Q demand, each edge $e = (u, v)$ must be traversed by enough tours to cover $d(T_v)$ demand. We call this value the *traffic* on the edge e, denoted $f(e)$. Namely, $f(e) = \lceil \frac{d(T_v)}{Q} \rceil$ tours. Each such tour traverses the edge exactly twice (once in each direction). We say that the lower bound $LB(e)$ of the contribution of edge $e = (u, v)$ to the total solution cost is therefore,

$$LB(e) = 2 \cdot l(e) \cdot f(e) = 2 \cdot l(e) \lceil \frac{d(T_v)}{Q} \rceil$$

and that the lower bound LB on OPT is

$$LB = \sum_{e \in E} LB(e)$$

For convenience, we scale down all demand values by a factor of Q and set $Q = 1$. We also assume that the vertices with positive demand are exactly the leaves: If some internal vertex v has positive demand $d(v)$ we can add a vertex v' with demand $d(v') = d(v)$ and edge (v, v') of length zero and set $d(v)$ to zero. Alternatively, if some leaf v has zero demand, no tour in an optimal solution will visit v, so v and the edge to v's parent can be deleted from the graph. Finally, we assume that no non-root vertex has degree exactly two, as no branching would occur at such a vertex, so the two incident edges can be spliced into one.

A very high level description of the algorithm is as follows: iteratively identify sets of tours in which the ratio of the cost of the tours to the reduction in cost to the lower bound, LB, is at most 4/3. We call such a set of tours a 4/3-*approximate tour set*.

We say that a 4/3-approximate tour set *removes* the demand that the tours cover. If such a tour set removes all demand from a branch, we say that this branch has been *resolved*. After a branch is resolved, it is convenient to think of it as having been deleted from the tree and proceed with the smaller instance.

We note that any 4/3-approximation algorithm for Capacitated Vehicle Routing trivially generates a 4/3-approximate tour set. The converse, is also straightforward:

▶ **Lemma 3.** *If, after iteratively finding and removing demand from 4/3-approximate tour sets, no demand remains, then the union of all tour sets is a 4/3-approximation for* Capacitated Vehicle Routing.

Given this, we make one more simplifying assumption that each leaf v has demand $d(v) < 1$. Assume to the contrary that for some leaf v, $d(v) \geq 1$. A tour that goes directly from r to v and back and covers one unit of demand at v is in fact a 1-approximate tour (and thus also a 4/3-approximate tour). If ever such a leaf exists, we can greedily take such tours until no more such leaves exist [2].

2.2 Safe Operations

We say that an operation that modifies the graph is *safe* if it does not decrease OPT or change the cost of the lower bound LB and it preserves feasibility. Note that a 4/3-approximate tour set in the modified graph is therefore also a 4/3-approximate tour set in the unmodified graph.

The algorithm proceeds iteratively. In each iteration, the algorithm performs a series of *safe* operations and takes a sequence of 4/3-approximate tours.

We now define the operations (see Appendix A for proofs of safeness). Note that our operations and simplifying assumptions generalize the operations that appear in [2] with different vocabulary (see Appendix B for a comparison).

- **Condense**: If edge $e = (u, v)$ has traffic $f(e) = 1$ and v is not a leaf, add a vertex v' and replace e and T_v with an edge $e' = (u, v')$ with $l(e') = l(e) + \sum_{e'' \in T_v} l(e'')$ and set $d(v') = d(T_v)$ (see Figure 1a).
- **Unzip**: If edge $e = (u, v)$ has traffic equal to the sum of the traffic on child edges $(v, w_1), (v, w_2), ..., (v, w_k)$, then delete v and add edges $(u, w_1), (u, w_2), ..., (u, w_k)$ with lengths $l((u, w_i)) = l(e) + l(v, w_i)$ for all $i \in \{1, 2, ..., k\}$ (see Figure 1b).
- **Group**: If vertex u has at least four children, including three leaf children v_1, v_2, v_3, such that $1.5 < d(v_1) + d(v_2) + d(v_3) < 2$, add vertex u', edge (u, u') of length zero, and for $i \in 1, 2, 3$, replace (u, v_i) with (u', v_i) (see Figure 1c).
- **Unite**: If vertex u has leaf children v_1 and v_2 such that $d(v_1) + d(v_2) \leq 1$, delete v_1 and v_2 and add vertex v_0 with demand $d(v_1) + d(v_2)$ and edge (u, v_0) with length $l((u, v_1)) + l((u, v_2))$ (see Figure 1d).
- **Slide**: If edge $e_0 = (u, v)$ has child edges $e_1 = (v, w_1)$ and $e_2 = (v, w_2)$ such that traffic $f(e_0) = f(e_1)$, then delete edge e_2 and add edge (w_1, w_2) of length $l(e_2)$ (see Figure 1e).

3 Algorithm

Exhaustively applying safe operations is called *simplifying* the instance. Note that none of the operations cancel each other, so this process terminates.

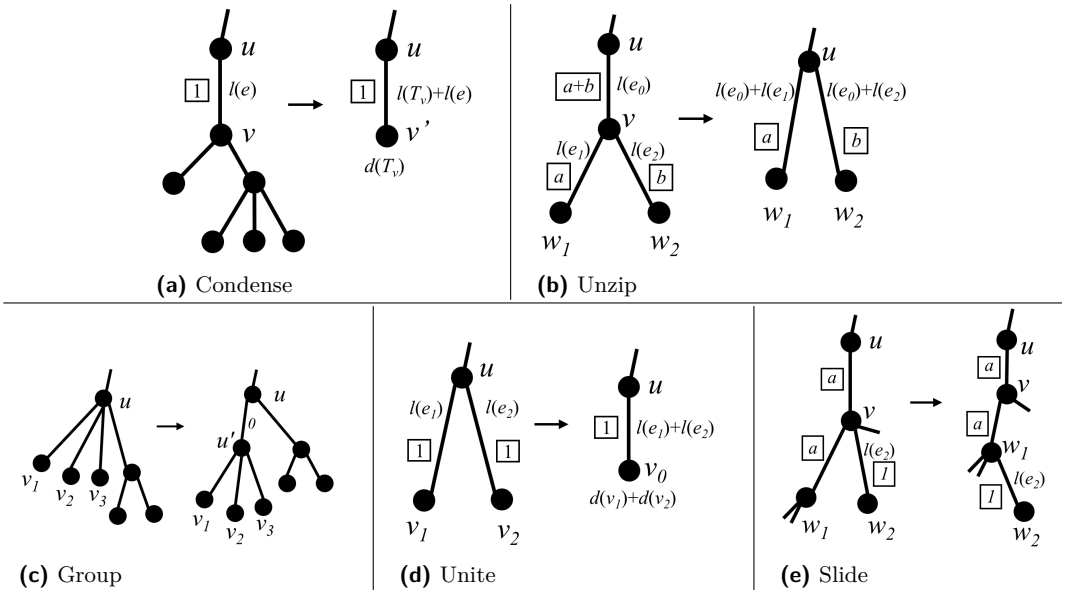


Figure 1 Safe Operations. Traffic values are shown in rectangles.

We say that a problem instance is *simplified* if no more safe operations are available, no internal vertices have demand, no non-root vertex has degree two, and for every leaf v, $0 < d(v) < 1$. We say that a branch is simplified if these conditions hold for the branch.

Recall that a *branch* consists of a subtree along with its parent edge (*stem*). If the branch has traffic p, we call it a *p-branch*.

A simplified 2-branch with stem $e_0 = (u, v)$ such that v has exactly three children w_1, w_2, w_3, all of which are leafs and such that $1.5 < d(w_1) + d(w_2) + d(w_3) \leq 2$ is called a 2-*chain*.

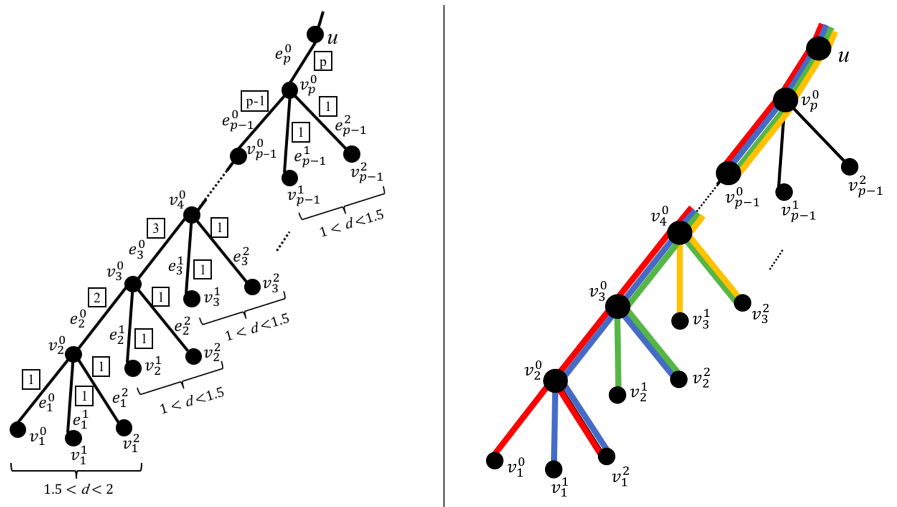
▶ **Lemma 4.** *In a simplified problem instance, all 2-branches are 2-chains.*

Proof. Consider any 2-branch in a simplified problem instance. Clearly no edge in the branch can have traffic greater than two. If more than one child edge of the stem had traffic two, then the traffic of the stem itself must be greater than two. If the stem had exactly one child edge with traffic two, then a slide operation would have been possible. Therefore every child edge of the stem has traffic one. Each of these edges must be leaf edges or else they could be condensed. If there were exactly two such edges, then the stem could be unzipped. Since no two of these edges can be united, then the demand of every pair sums to more than one, so there are exactly three such edges and their demand sums to more than 1.5. ◀

3.1 p-Chains

We now generalize the notion of a 2-chain. For $p \geq 3$ a *p-chain* is a simplified p-branch with stem $e_0 = (u, v)$ such that v has exactly three children w_1, w_2, w_3, in which w_2 and w_3 are leaves with $1 < d(w_2) + d(w_3) \leq 1.5$ and (v, w_1) is the stem of a $(p-1)$-chain (see Figure 2a).

For convenience, we define a labeling scheme for p-chains. Each vertex v_i^j is doubly indexed by level i and rank (child-order) j. We use e_i^j to denote the parent edge of v_i^j, $\forall i, j$. A 2-chain with parent u has stem $e_2^0 = (u, v_2^0)$ leaves v_1^0, v_1^1, and v_1^2 such that $l(e_1^0) \geq l(e_1^1) \geq l(e_1^2)$. For $p > 2$, a p-chain with parent u has stem $e_p^0 = (u, v_p^0)$, and children v_{p-1}^0, v_{p-1}^1, and v_{p-1}^2, such that e_{p-1}^0 is the stem of a $p-1$-chain, and v_{p-1}^1 and v_{p-1}^2 are leaves with $l(e_{p-1}^1) \geq l(e_{p-1}^2)$ (See Figure 2a).



(a) p-Chain. Traffic values are shown in rectangles, and d denotes demand.

(b) Cascade. The first four tours of a cascade tour set are depicted.

Figure 2

We further classify some p-chains as *long* p-chains: All 2-chains are long, and for $p \geq 3$ a *long* p*-chain* is a p-chain in which $l(e_{p-1}^2) < l(P[v_p^0, r])$ and e_{p-1}^0 is the stem of a *long* $(p-1)$-chain. A p-chain in which $l(e_{p-1}^2) \geq l(P[v_p^0, r])$ is called a *short* p-chain.

Long p-chains are particularly convenient because they can be *resolved* individually at the root and in sibling pairs for internal vertices, as described in the following lemmas (which we prove in Section 5).

▶ **Lemma 5.** *Long p-chains can be resolved at the root.*

▶ **Lemma 6.** *A long p-chain and long p'-chain can be resolved together if they are sibling branches.*

As in [2], our algorithm proceeds in a series of iterations. Each iteration performs a set of safe operations and identifies a 4/3-approximate tour set.

▶ **Lemma 7.** *Iteration i runs in polynomial time and either finds a nonempty 4/3-approximate tour set or finds that every branch at the root is either a long p-chain or 1-branch.*

Proof. See Section 4. ◀

These iterations continue until every branch at the root is either a long p-chain or 1-branch. The algorithm then solves the remaining instance, using the result from Lemma 8.

▶ **Lemma 8.** *There is a polynomial time 4/3-approximation algorithm for instances of* Capacitated Vehicle Routing *on trees in which every child branch of the root is either a long p-chain or a 1-branch.*

Proof. The cost of a tour that traverses a 1-branch at the root is equivalent to the reduction to lower bound LB that results from removing the demand of the branch, so such a tour is a 1-approximate tour (and thus also a 4/3-approximate tour). The lemma result then follows from Lemma 3 and Lemma 5 ◀

Putting these steps together, gives our overall 4/3-approximation result described in Theorem 2.

▶ **Theorem 2.** *There is a polynomial-time 4/3 approximation for* Capacitated Vehicle Routing *in trees that is tight with respect to LB.*

Proof. Let m denote the total amount of demand in the graph. Since each iteration removes demand, there are at most m iterations, each of which runs in polynomial time. By Lemma 7, after iteration m, every branch off the root must be a long p-chain or a 1-branch. By Lemma 8 there is a polynomial-time 4/3-approximation for these branches. Combining this approximation with the collection of tours identified during the iterations results in an overall 4/3 approximation, by Lemma 3. ◀

All that remains is to prove the above lemmas. In Section 4 we describe the iteration subroutine and prove Lemma 7. In Section 5 we prove Lemmas 5 and 6.

4 Description of Iteration i

We say that a p-branch B is *settled* if it is either a 1-branch or a long p-chain. Otherwise we say that B is *unsettled*. We say that a p-branch B is *minimally unsettled* if it is unsettled and all of its child branches are settled.

Each iteration consists of the following subroutine:

1. Simplify the instance (i.e. exhaustively apply safe operations)
2. If all branches at the root are settled, terminate iteration.
3. Otherwise, find a minimally unsettled branch B.
   - **(i)** If B has at least two child branches that are long p-chains, apply Lemma 6.
   - **(ii)** Otherwise, if B has at least three child branches that are 1-branches, apply Lemma 9
   - **(iii)** Otherwise, B is a (short) p-chain. Apply Lemma 10.

▶ **Lemma 9.** *A simplified, minimally unsettled branch B with at least three child branches that are 1-branches admits a 4/3-approximate tour set.*

Proof. Let (u, v_0) be the stem of B, and let (v_0, v_1), (v_0, v_2), and (v_0, v_3) be three (child) 1-branches. Since the branch is simplified, v_1, v_2, and v_3 are leaves. Since the unite operation is unavailable, $d(v_1) + d(v_2) + d(v_3) > 1.5$, and since B is unsettled, it is not a 2-chain. Furthermore, the group operation is unavailable, so $2 < d(v_1) + d(v_2) + d(v_3) < 3$. Let $a = l(P[v_0, r])$, $w_1 = l(v_0, v_1)$, $w_2 = l(v_0, v_2)$, and $w_3 = l(v_0, v_3)$. Without loss of generality, assume $w_1 \leq w_2 \leq w_3$.

If $a \leq w_1 + w_2 + w_3$, then for $i \in \{1, 2, 3\}$, let t_i be the tour that travels from the depot to v_i, covers all demand at v_i, and then returns to the depot. The length of t_i is $2(a + w_i)$. The total cost of the tour set $\{t_1, t_2, t_3\}$ is $2(3a + w_1 + w_2 + w_3)$. This tour set covers all demand of the leaves and also reduces demand along $P[v_0, r]$ by two, since $2 < d(v_1) + d(v_2) + d(v_3) < 3$, so the reduction to LB is $2(2a + w_1 + w_2 + w_3)$. The ratio of the cost of the tour set to the reduction to the cost of LB that results from taking these tours is therefore,

$$\frac{2(3a + w_1 + w_2 + w_3)}{2(2a + w_1 + w_2 + w_3)} = 1 + \frac{a}{2a + w_1 + w_2 + w_3} \leq \frac{4}{3}$$

where the final equality comes from $a \leq w_1 + w_2 + w_3$.

Otherwise, $a > w_1 + w_2 + w_3$. Let t be the tour that travels from the depot to v_1, covers all demand at v_1, travels to v_3, covers as much demand as possible at v_3, and then returns to the depot. The cost of t is $2(a + w_1 + w_3)$. Note that since the unite operation is unavailable, then $d(v_1) + d(v_3) > 1$, so the vehicle is full, and some demand remains at v_3. Since the vehicle is full, then t reduces demand along $P[v_0, r]$ by one, so the reduction to

LB is $2(w_1 + a)$. The ratio of the cost of the tour set $\{t\}$ to the reduction to the cost of LB that results from taking these tours is therefore,

$$\frac{2(a + w_1 + w_3)}{2(w_1 + a)} = 1 + \frac{w_3}{w_1 + a} \leq \frac{4}{3}$$

where the final inequality comes from $a > w_1 + w_2 + w_3 \geq 3w_3$. ◀

▶ **Lemma 10.** *A short p-chain admits a 4/3-approximate tour set.*

Proof. Let $a = l(P[v_p^0, r])$, $b = l(e_{p-1}^2)$, and $c = l(e_{p-1}^1)$. By construction, $c \geq b$, and since the p-chain is short, $b \geq a$. Let t_1 be the tour that goes from the depot to v_{p-1}^1, covers all demand at this leaf, and then returns to the depot. Similarly, let t_2 be the tour that goes from the depot to v_{p-1}^2, covers all demand at this leaf, and then returns to the depot. Tour t_1 has cost $2(a + c)$ and t_2 has cost $2(a + b)$. Tour set $\{t_1, t_2\}$ has total cost $2(2a + b + c)$. This tour set covers all demand at these leaves, which sum to greater than one by definition of p-chain, so these tours reduce the demand along $P[v_p^0, r]$ by one. Therefore the reduction to LB from this tour set is $2(a + b + c)$. Therefore the ratio of the cost of the tour set to the reduction to the cost of LB that results from taking these tours is,

$$\frac{2(2a + b + c)}{2(a + b + c)} = 1 + \frac{a}{a + b + c} \leq \frac{4}{3}$$

The final inequality comes from $a \leq b \leq c$. ◀

Finally, we prove Lemma 7, which we restate here for convenience.

▶ **Lemma 7.** *Iteration i runs in polynomial time and either finds a nonempty 4/3-approximate tour set or finds that every branch at the root is either a long p-chain or 1-branch.*

Proof. To prove this lemma, we first must show that any minimally unsettled branch B in a simplified instance must be in at least one of the three identified cases. Let B be such a branch. B must have child branches, since it is unsettled. Since it is minimally unsettled, every child branch is settled (i.e. either a 1-branch or a long p-chain).

If B has at least two long p-chains as child branches, then case i holds.

If B has no long p-chains as child branches, then all child branches are 1-branches. By Lemma 4, in a simplified instance all 2-branches are 2-chains. Since all 2-chains are long, and thus settled, B must be a j-branch for some $j \geq 3$. Since the unzip operation is unavailable, then there are at least four 1-branches as child branches of B, so case ii holds.

If B has exactly one long p-chain as a child branch and case ii doesn't hold, then B must be a short p'-chain. If there were no 1-branches as child branches, then the degree-two vertex could be spliced. If there were exactly one 1-branch then either $p = p'$ and a slide operation would be available, or $p = p' - 1$ and an unzip operation would be available. Since case ii doesn't hold, there are exactly two 1-branches as child branches. Since an unzip operation is unavailable, $p = p' - 1$, and since a unite operation is unavailable, the demand of the two 1-branches (which are both leaves by the condense operation) sum to greater than one. Since the instance is simplified, then B is a p'-chain. But since B is unsettled, it must be a short p'-chain. Therefore case iii holds.

Since B is covered by some case, then the corresponding lemma (Lemma 6, 9, or 10) guarantees a nonempty 4/3-approximate tour set.

If no unsettled branch can be found, then all branches at the root must be settled.

Finally, to see that the iteration runs in polynomial time, note that each operation can be performed in constant time. Condense, unite, unzip, (and splicing) all make the

graph smaller, so they are exhausted in linear time. Subsequently, if condense and unite are unavailable, then each slide results in a degree two vertex remaining to be spliced, also making the graph smaller. After all other operations are exhausted, each leaf can participate in at most one group operation in a given iteration. Each of the three cases can likewise be handled in linear time. ◀

5 Resolving p-Chains

5.1 Cascades

In order to prove Lemmas 5 and 6 we describe a specific group of tours that collectively covers the demand of a long p-chain.

The labeling on p-chains gives a natural bottom-up ordering on the leaves: $v_1^0, v_1^1, v_1^2, v_2^1, v_2^2$, ..., v_{p-1}^1, v_{p-1}^2, where the order is determined first by level and then by rank. For a long p-chain, we define a *cascade* to be the sequence of p tours in which each tour considers the leaves in this bottom-up order and:

1. Visits and covers all demand from the first leaf with remaining demand.
2. While the vehicle has spare capacity and there is unmet demand in the branch, determines the lowest level i with leaves with unmet demand and covers as much demand as possible from v_i^2.
3. Returns to the depot.

Let the resulting tours be $t_1, ..., t_p$. Tour t_1 first covers all demand from v_1^0 and then covers $1 - d(v_1^0)$ demand from v_1^2. Since by definition, a p-chain is simplified, the unite operation is unavailable, so $d(v_1^0) + d(v_1^2) > 1$, implying the vehicle is full. The second tour t_2 covers all demand from v_1^1, covers all remaining demand at v_1^2 (since $d(v_1^0)+d(v_1^1)+d(v_1^2) < 2$) and covers some demand at v_2^2. Since the slide operation is unavailable, $d(v_1^0)+d(v_1^1)+d(v_1^2)+d(v_2^2) > 3$, so the vehicle is full. Note that both vehicles have been full, and after two tours no demand remains at level 1. This pattern continues. Tour t_i for $3 \leq i < p$ covers all demand at v_{i-1}^1, all remaining demand at v_{i-1}^2, and some demand at v_i^2. Inductively, since slide operation is unavailable and all previous vehicles have been full, this vehicle must also be full. After t_i, no demand remains below level i. Since all vehicles have been full, there is less than one unit of demand remaining for t_p (and no demand remains below level $p-1$), so t_p covers all demand from v_{p-1}^1 and all remaining demand from v_{p-1}^2. (See Figure 2b).

This cascading pattern results in i tours traversing e_i^0, one tour traversing e_i^1, two tours traversing e_i^2 for all i, and p tours traversing all edges in the path from the depot r to v_p^0. Note in particular that these values match the lower bounds given by the edge traffic, for e_i^0 and e_i^1. The excess cost, therefore, comes from doubling the traffic lower bound on the e_i^2 edges as well as extra traffic along the path to the depot.

5.2 Proofs of Lemmas 5 and 6

We restate the lemmas here for convenience. Recall that a branch is *resolved* if all of its demand is covered by a 4/3-approximate tour group. That is, a set of tours whose total cost is at most 4/3 times the reduction to the lower bound LB that results from removing the demand of the branch.

▶ **Lemma 5.** *Long p-chains can be resolved at the root.*

Proof. Let B be a long p-chain at the root (depot). Consider the cascade on B. As described in Section 5.1, the cost of the p tours in the cascade is

$$2\big(p \cdot l(e_p^0) + \sum_{i=1}^{p-1} i \cdot l(e_i^0) + 2l(e_i^2) + l(e_i^1)\big)$$

since the cost of the path to the depot is zero. Additionally the cascade covers all demand in B, so the reduction to LB after covering B is:

$$\sum_{e \in B} LB(e) = \sum_{e=(u,v) \in B} 2 \cdot l(e) \cdot \lceil d(T_v) \rceil = 2\big(p \cdot l(e_p^0) + \sum_{i=1}^{p-1} i \cdot l(e_i^0) + l(e_i^2) + l(e_i^1)\big)$$

Taking the ratio of cost to reduction in cost of LB gives our result:

$$\frac{2\big(p \cdot l(e_p^0) + \sum_{i=1}^{p-1} i \cdot l(e_i^0) + 2l(e_i^2) + l(e_i^1)\big)}{2\big(p \cdot l(e_p^0) + \sum_{i=1}^{p-1} i \cdot l(e_i^0) + l(e_i^2) + l(e_i^1)\big)} = 1 + \frac{\sum_{i=1}^{p-1} l(e_i^2)}{p \cdot l(e_p^0) + \sum_{i=1}^{p-1} i \cdot l(e_i^0) + l(e_i^2) + l(e_i^1)}$$

$$\leq 1 + \frac{\big(l(e_1^2)\big) + \big(\sum_{i=2}^{p-1} l(e_i^2)\big)}{\big(l(e_1^0) + l(e_1^1) + l(e_1^2)\big) + \big(\sum_{i=2}^{p-1} l(e_i^2) + l(e_i^1) + \sum_{j=i+1}^{p} l(e_j^0)\big)}$$

$$\leq 1 + \frac{\big(l(e_1^2)\big) + \big(\sum_{i=2}^{p-1} l(e_i^2)\big)}{\big(3 \cdot l(e_1^2)\big) + \big(\sum_{i=2}^{p-1} 3 \cdot l(e_i^2)\big)} \leq \frac{4}{3}$$

Where the second to last inequality comes from the fact that because B is a long p-chain, $l(e_i^2) < l(P[v_{i+1}^0, r]) = \sum_{j=i+1}^{p} l(e_j^0)$, and because $l(e_i^2) \leq l(e_i^1)$ by construction. ◀

▶ **Lemma 6.** *A long p-chain and long p'-chain can be resolved together if they are sibling branches.*

Proof. Let u be a vertex with child branches B a long p-chain and B' a long p'-chain. Consider the cascades on B and B'. The cost of these $p + p'$ tours is

$$2\Big[\big(p \cdot l(e_p^0) + \sum_{i=1}^{p-1} i \cdot l(e_i^0) + 2l(e_i^2) + l(e_i^1)\big) + \big(p' \cdot l(e'^0_{p'}) + \sum_{i=1}^{p'-1} i \cdot l(e'^0_i) + 2l(e'^2_i) + l(e'^1_i)\big) + (p + p')l(P[u,r])\Big]$$

Additionally the cascade covers all demand in B and B', so the reduction to LB is $\sum_{e \in B \cup B'} LB(e) = \sum_{e \in B \cup B'} 2 \cdot l(e) \cdot f(e)$, which is

$$2\Big[\big(p \cdot l(e_p^0) + \sum_{i=1}^{p-1} i \cdot l(e_i^0) + l(e_i^2) + l(e_i^1)\big) + \big(p' \cdot l(e'^0_{p'}) + \sum_{i=1}^{p'-1} i \cdot l(e'^0_i) + l(e'^2_i) + l(e'^1_i)\big) + (p + p' - 1)l(P[u,r])\Big]$$

Note that the $p + p' - 1$ factor in the reduction to the edges along $P[u, r]$ arises because by definition the total demand in a p-chain is between $p - 0.5$ and p, so covering all demand

in B and B' reduces the demand in T_u by at least $p + p' - 1$. Rearranging the terms, this reduction to the lower bound is greater than:

$$2\Big[\Big(\sum_{i=0}^{2} l(e_1^i) + \sum_{i=2}^{p-1} l(P[v_{i+1}^0, r]) + l(e_i^2) + l(e_i^1)\Big)$$

$$+\Big(\sum_{i=0}^{2} l(e'^i_1) + \sum_{i=2}^{p'-1} l(P[v'^0_{i+1}, r]) + l(e'^2_i) + l(e'^1_i)\Big) + 3l(P[u,r])\Big]$$

$$= 2(X + Y + X' + Y' + Z)$$

Where

$$X = \sum_{i=0}^{2} l(e_1^i)$$

$$Y = \sum_{i=2}^{p-1} l(P[v_{i+1}^0, r]) + l(e_i^2) + l(e_i^1)$$

$$X' = \sum_{i=0}^{2} l(e'^i_1)$$

$$Y' = \sum_{i=2}^{p'-1} l(P[v'^0_{i+1}, r]) + l(e'^2_i) + l(e'^1_i)$$

and

$$Z = 3l(P[u,r]).$$

The ratio of cost to reduction in cost of LB is therefore at most,

$$1 + \frac{\sum_{i=1}^{p-1} l(e_i^2) + \sum_{i=1}^{p'-1} l(e'^2_i) + l(P[u,r])}{X + Y + X' + Y' + Z} \leq \frac{4}{3}$$

Where the final inequality comes from noting that, by construction, $l(e_1^2) \leq X/3$ and $l(e'^2_1) \leq X'/3$, and by definition of a long p-chain, $l(e_i^2) \leq \min\{l(e_i^1), l(e_i^2), l(P[v_{i+1}^0, r])\}$ for all $2 \leq i < p$, so $\sum_{i=2}^{p-1} l(e_i^2) \leq Y/3$ and $\sum_{i=2}^{p-1} l(e'^2_i) \leq Y'/3$.

◀

References

1. Kemal Altinkemer and Bezalel Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6(4):149–158, 1987.
2. 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.
3. 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. ACM, 1997.

4 Amariah Becker, Philip N Klein, and David Saulpic. Polynomial-time approximation schemes for k-center and bounded-capacity vehicle routing in metrics with bounded highway dimension. *arXiv preprint arXiv:1707.08270*, 2017.

5 Amariah Becker, Philip N Klein, and David Saulpic. A quasi-polynomial-time approximation scheme for vehicle routing on planar and bounded-genus graphs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 87. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

6 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

7 Aparna Das and Claire Mathieu. A quasi-polynomial time approximation scheme for Euclidean capacitated vehicle routing. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 390–403. SIAM, 2010.

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

9 Mark Haimovich and AHG Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of operations Research*, 10(4):527–542, 1985.

10 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.

11 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.

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

13 Christos H Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

A Showing That Safe Operations Are Safe

In this section we show that safe operations are actually safe. Recall that an operation that modifies the graph is *safe* if it does not decrease OPT or change the cost of the lower bound LB and it preserves feasibility.

The **condense** operation replaces a branch B in which every edge e has traffic $f(e) = 1$ with a single edge e_B of length $\sum_{e \in B} l(e)$ and traffic $f(e_B) = 1$, so the lower bound LB remains unchanged, since the branch contributes $2 \cdot \sum_{e \in B} l(e) \cdot 1 = 2 \cdot l(e_B) \cdot 1$ toward LB both before and after the operation. The operation is equivalent to requiring any tour that covers *any* client demand in B to traverse *all of* B, which clearly preserves feasibility and possibly *increases*, but cannot decrease, OPT.

Before the **unzip** operation, the contribution to LB of edges involved in the modification is $2 \cdot l(e) \cdot \left(\sum_{i=1}^{k} f((v, w_i)) \right) + \sum_{i=1}^{k} 2 \cdot l((v, w_i)) \cdot f((v, w_i))$ which equals $\sum_{i=1}^{k} 2 \cdot (l(e) + l((v, w_i))) \cdot f((v, w_i))$, the contribution to LB of involved edges *after* the operation. The length and traffic values (and thus also the contribution to LB) of all other edges remain unchanged. The operation is equivalent to requiring every tour that traverses c child edges of e to traverse e a total of $2c$ times. This redundancy may increase the value of OPT but cannot decrease it. Since any tour can be extended to cover e multiple times, the operation also preserves feasibility.

The **group** operation simply inserts an edge of length zero, resulting in an equivalent instance, and clearly has no affect on LB, OPT, or feasibility. The operation is merely for convenience.

The **unite** operation is equivalent to first inserting an edge of length zero (and traffic one) and then performing a condense operation, both of which are shown above to be safe. The composition of safe operations results in a safe operation.

The **slide** operation essentially *moves* an edge without changing the length or traffic values of any edge, which clearly preserves the value of LB. The operation is equivalent to requiring every tour that traverses edge e_2 to *also* traverse edge e_1, which may increase OPT, but cannot decrease OPT. Since the tree is connected, any tour can be extended to include e_1, so feasibility is preserved.

B Reforming Operations from Asano et al. [2]

Asano et al. introduce seven safe *Reforming Operations* [2]. In this section we compare these to the safe operations we use in this paper.

Reforming operation R_1 greedily takes a tour at full capacity to any vertex with demand greater than one. After exhaustively applying R_1, no vertex has demand greater than one. Reforming operation R_2 moves any demand at internal nodes to leaves, by adding edges of length zero when necessary.

In this paper, we use simplifying assumptions to accomplish the same result as exhaustive application of these two operations. That is, we assume that the input graph already has the property that leaves are the only vertices with demand, and that the demand of every leaf is at most one. Our algorithm does not also need do use these operations, because these properties of the demand are never *undone* by our algorithm.

Reforming operation R_3 addresses the specific case of a vertex u with a leaf vertex v of demand at most one as its only child, by contracting the edge (u, v), increasing $l((p(u), u))$ by $l((u, v))$, and setting $d(u) = d(v)$. Reforming operation R_4 similarly contracts an entire subtree with total demand at most one to a single leaf. In this paper, our **condense** operation generalizes both R_3 and R_4 as it operates on branches rather than subtrees. Note that R_3 also addresses the case where $1 < d(u) + d(v) \leq 2$, but this case never arises in our algorithm because of our simplifying assumptions.

Reforming operation R_5 merges leaves if their combined demand is at most one *and* if the parent vertex u of the leaves has no other child v such that the subtree rooted at v has total demand at least two. In this paper, our **unite** operation generalizes R_5.

Reforming operation R_6 addresses the case where a vertex v has exactly two children v_1 and v_2 that are leaves such that $1 < d(v_1) + d(v_2) < 2$. If v has parent u, R_6 removes v and adds edge (u, v_1) of weight $l((u, v)) + l((v, v_1))$ and edge (u, v_2) of weight $l((u, v)) + l((v, v_2))$. Our **unzip** operation generalizes R_6 to accommodate more than two children, non-leaf children, and larger demands.

Finally, reforming operation R_7 addresses the case where a vertex v has children v_1 and v_2 such that v_2 is a leaf and the subtree rooted at v_1 has total demand d_1 such that $1 < d_1 < 2$ *and* such that the subtree rooted at v has total demand d such that $1 < d < 2$. R_7 replaces edge (v, v_2) with edge (v_1, v_2) of length $l((v, v_2))$. Our **slide** operation generalizes R_7 to accommodate larger demand values.

C Example Showing Tightness

In this section we present the example of a tree instance that has cost at least $4/3 \cdot LB$ that was given by Asano et al. [2].

They define a tree T with root r, in which r has a single child q, and q has $2n+1$ children $v_1, v_2, ..., v_{2n+1}$ that are all leaves. Edge (r, q) as well as edges (q, v_i) for all $1 \leq i \leq 2n+1$ have length one. That is, T is a tree of height two in which all edges have unit lengths. Furthermore, the demand of each leaf v_i is $0.5 + \epsilon$, where $\epsilon < 1/(4n+2)$.

For this instance, the traffic of every edge (q, v_i) is one, and the traffic of edge (r, q) is $n+1$, so $LB = 2 \cdot 1 \cdot (n+1) + (2n+1) \cdot 2 \cdot 1 \cdot 1 = 2n + 2 + 4n + 2 = 6n + 4$.

An optimum solution for this instance consists of $2n+1$ tours $t_1, t_2, ..., t_{2n+1}$ in which tour t_i goes from r to v_i, covers all demand at v_i and then immediately returns to the depot.

The cost OPT of this optimum solution is $OPT = (2n+1) \cdot 4 = 8n + 4$.

The ratio of OPT to LB is $\frac{8n+4}{6n+4}$ which tends to $4/3$ as n goes to infinity [2].

Note that there are in fact many other optimum solutions for this instance. For example consider the following solution consisting of $n+1$ tours $t_1, t_2, ..., t_{n+1}$. For $i \in \{1, 2, ..., n\}$, t_i goes from r to v_{2i-1}, covers all demand at v_{2i-1}, then covers as much demand at v_{2i} as possible before returning to r. Tour t_{n+1} then covers all demand at v_{2n+1} and then all remaining demand at leaves v_{2i} for $i \in \{1, 2, ..., n\}$. The cost of this solution is $2 \cdot n \cdot 3 + 2 \cdot 1 \cdot (n+2) = 6n + 2n + 4 = 8n + 4$.