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

Aditya Jayaprakash
Department of Computing Science
University of Alberta
jayaprak@ualberta.ca

Mohammad R. Salavatipour[†]
Department of Computing Science
University of Alberta
mrs@ualberta.ca

Abstract

In this paper, we present Approximation Schemes for Capacitated Vehicle Routing Problem (CVRP) on several classes of graphs. In CVRP, introduced by Dantzig and Ramser in 1959 [13], we are given a graph $G = (V, E)$ with metric edges costs, a depot $r \in V$, and a vehicle of bounded capacity Q. The goal is to find a minimum cost collection of tours for the vehicle that returns to the depot, each visiting at most Q nodes, such that they cover all the nodes. This generalizes classic TSP and has been studied extensively. In the more general setting, each node v has a demand d_v and the total demand of each tour must be no more than Q. Either the demand of each node must be served by one tour (unsplittable) or can be served by multiple tours (splittable). The best known approximation algorithm for general graphs has ratio $\alpha + 2(1-\epsilon)$ (for the unsplittable) and $\alpha + 1 - \epsilon$ (for the splittable) for some fixed $\epsilon > \frac{1}{3000}$, where α is the best approximation for TSP. Even for the case of trees, the best approximation ratio is 4/3 [5], and it has been an open question if there is an approximation scheme for this simple class of graphs. Das and Mathieu [14] presented an approximation scheme with time $n^{\log^{O(1/\epsilon)} n}$ for Euclidean plane $\mathbb{R}^2$. No other approximation scheme is known for any other class of metrics (without further restrictions on Q). In this paper, we make significant progress on this classic problem by presenting Quasi-Polynomial Time Approximation Schemes (QPTAS) for graphs of bounded treewidth, graphs of bounded highway dimensions, and graphs of bounded doubling dimensions. For comparison, our result implies an approximation scheme for Euclidean plane with run time $n^{O(\log^6 n/\epsilon^5)}$.

1 Introduction

Vehicle routing problems (VRP) describe a class of problems where the objective is to find cost-efficient delivery routes for delivering items from depots to clients using vehicles having limited capacity. These problems have numerous applications in real-world settings. The Capacitated Vehicle Routing Problem (CVRP) was introduced by Dantzig and Ramser in 1959 [13]. In CVRP, we are given as input a graph $G = (V, E)$ with metric edge weights (also referred to as costs) $w(e) \in \mathbb{Z}^{\geq 0}$, a depot $r \in V$, along with a vehicle of capacity $Q > 0$, and wish to compute a minimum weight/cost collection of tours, each starting from the depot and visiting at most Q customers, whose union covers all the customers. In the more general setting, each node v has a demand $d(v) \in \mathbb{Z}^{\geq 1}$ and the goal is to find a set of tours of the minimum total cost, each of which includes r such that the union of the tours covers the demand at every client and every tour covers at most Q demand.

There are three common versions of CVRP: *unit*, *splittable*, and *unsplittable*. In the splittable variant, the demand of a node can be delivered using multiple tours, but in the unsplittable variant, the entire demand of a client must be delivered by a single tour. The unit demand case is a special case of the unsplittable case where every node has unit demand, and the demand of a client must be delivered by a single tour. CVRP has also been referred to as the k-tours problem [3, 4]. All three variants admit constant factor approximation algorithm in polynomial-time [17]. Haimovich et al. [17] showed that a heuristic called iterative partitioning (which starts from a TSP tour and breaks the tour into capacity respecting tours by making a trip back and forth to the depot) implies an $(\alpha + 1(1 - 1/Q))$-approximation for the unit demand case, with α being the approximation ratio of Traveling Salesman Problem (TSP). A similar approach implies a $2 + (1 - 2/Q)\alpha)$-approximation for the

[*]The full version of the paper can be accessed at `https://arxiv.org/abs/2106.15034`

[†]Supported by NSERC.

unsplittable variant [2]. Very recently, Blauth et al. [10] improved these approximations by showing that there is an $\epsilon > 0$ such that there is an $(\alpha + 2 \cdot (1 - \epsilon))$-approximation algorithm for unsplittable CVRP and a $(\alpha + 1 - \epsilon)$-approximation algorithm for unit demand CVRP and splittable CVRP. For $\alpha = 3/2$, they showed $\epsilon > 1/3000$. All three variants are APX-hard in general metric spaces [24], so a natural research focus has been on structured metric spaces, i.e. special graph classes. Even on trees (and in particular on stars) CVRP remains NP-hard [22], and there exist constant-factor approximations (currently being 4/3 [5]), better than those for general metrics, however, the following question has remained open:

Question. Is it possible to design an approximation scheme for CVRP on trees, or more generally, graphs of bounded treewidth?

We answer the above question affirmatively. For ease of exposition, we start by proving the following first:

THEOREM 1.1. *For any $\epsilon > 0$, there is an algorithm that, for any instance of the unit demand CVRP on trees outputs a $(1+\epsilon)$-approximate solution in time $n^{O(\log^4 n/\epsilon^3)}$. For any instance of the splittable CVRP on trees when $Q = n^{O(\log^c n)}$ the algorithm runs in time $n^{O(\log^{2c+4} n)}$.*

We then show how this result can be extended to design QPTAS for graphs of bounded treewidth.

THEOREM 1.2. *For any $\epsilon > 0$, there is an algorithm that, for any instance of the unit demand CVRP on a graph G of bounded treewidth k outputs a $(1+\epsilon)$-approximate solution in time $n^{O(k^2 \log^3 n/\epsilon^2)}$. For the splittable CVRP on graphs of bounded treewidth when $Q = n^{O(\log^c n)}$, the algorithm outputs a $(1+\epsilon)$-approximate solution in time $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$.*

As a consequence of this and using earlier results of embedding graphs of bounded doubling dimensions or bounded highway dimensions into graphs of low treewidth, we obtain approximation schemes for CVRP on those graph classes.

THEOREM 1.3. *For any $\epsilon > 0$ and fixed $D > 0$, there is a an algorithm that, given an instance of the splittable CVRP with capacity $Q = n^{\log^c n}$ on a graph of doubling dimension D, finds a $(1+\epsilon)$-approximate solution in time $n^{O(D^D \log^{2c+D+3} n/\epsilon^{D+2})}$.*

As an immediate corollary, this implies an approximation scheme for CVRP on Euclidean metrics on $\mathbb{R}^2$ in time $n^{O(\log^6 n/\epsilon^5)}$ which improves on the run time of $n^{\log^{O(1/\epsilon)} n}$ of QPTAS of [14].

THEOREM 1.4. *For any $\epsilon > 0, \lambda > 0$ and $D > 0$, there is a an algorithm that, given a graph with highway dimension D with violation λ as an instance of the splittable CVRP with capacity $Q = n^{\log^c n}$, finds a solution whose cost is at most $(1+\epsilon)$ times the optimum in time $n^{O(\log^{2c+3+\log^2(\frac{D}{\epsilon\lambda}) \cdot \frac{1}{\lambda}} n/\epsilon^2)}$.*

1.1 Related Works

CVRP generalizes the classic TSP problem (with $Q = n$). For general metrics, Haimovich et al. [17] considered a simple heuristic, called tour partitioning, which starts from a TSP tour and then splits the tour into tours of size at most Q (by making back-and-forth trips to r) and showed that it is a $(1 + (1 - 1/Q)\alpha)$-approximation for splittable CVRP, where α is the approximation ratio for TSP. Essentially the same algorithm implies a $(2 + (1 - 2/Q)\alpha)$-approximation for unsplittable CVRP [2]. These stood as the best-known bounds until recently, when Blauth et al. [10] showed that given a TSP approximation α, there is an $\epsilon > 0$ such that there is an $(\alpha + 2 \cdot (1 - \epsilon))$-approximation algorithm for CVRP. For $\alpha = 3/2$, they showed $\epsilon > 1/3000$. They also showed a $(\alpha + 1 - \epsilon)$-approximation algorithm for unit demand CVRP and splittable CVRP.

For the case of trees, Labbé et al. [22] showed splittable CVRP is NP-hard, and Golden et al. [16] showed unsplittable version is APX-hard and hard to approximate better than 1.5. For splittable CVRP (again on trees), Hamaguchi et al. [18] defined a lower bound for the cost of the optimal solution and gave a 1.5 approximation with respect to the lower bound. Asano et al. [4] improved the approximation to $(\sqrt{41} - 1)/4$ with respect to the same lower bound and also showed the existence of instances whose optimal cost is exactly 4/3 times the lower bound. Becker [5] gave a 4/3-approximation with respect to the lower bound. Becker and Paul [9] showed a $(1, 1+\epsilon)$-bicriteria polynomial-time approximation scheme for splittable CVRP in trees, i.e. a PTAS but the capacity of every tour is up to $(1+\epsilon)Q$.

Das and Mathieu [14] gave a quasi-polynomial-time approximation scheme (QPTAS) for CVRP in the Euclidean plane ($\mathbb{R}^2$). A PTAS for when Q is $O(\log n / \log\log n)$ or Q is $\Omega(n)$ was shown by Asano et al.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

[4]. A PTAS for Euclidean plane $\mathbb{R}^2$ for all moderately large values of $Q \leq 2^{\log^{\delta} n}$, where $\delta = \delta(\epsilon)$, was shown by Adamaszek et al [1], building on the work of Das and Mathieu [14], and using it as a subroutine. For high dimensional Euclidean spaces $\mathbb{R}^d$, Khachay et al. [19] showed a PTAS when Q is $O(\log^{1/d} n)$. For graphs of bounded doubling dimension, Khachay et al. [20] gave a QPTAS when the number of tours is polylog(n) and Khachay et al. [21] gave a QPTAS when Q is polylog(n).

The following results are all for when Q is fixed. CVRP is APX-hard in general metrics and is polynomial-time solvable on trees. There exists a PTAS for CVRP in the Euclidean plane ($\mathbb{R}^2$) (again for when Q is fixed) as shown by Khachay et al. [19]. A PTAS for planar graphs was shown by Becker et al. [8] and a QPTAS for planar and bounded-genus graphs were shown by Becker et al. [6]. A PTAS for graphs of bounded highway dimension and an exact algorithm for graphs with treewidth with running time $O(n^{\text{tw}Q})$ was shown by Becker et al. [7]. Cohen-Addad et al. [12] showed an efficient PTAS for graphs of bounded-treewidth, an efficient PTAS for bounded highway dimension, an efficient PTAS for bounded genus metrics and a QPTAS for minor-free metrics. Again, note that these results are all under the assumption that Q is fixed.

So aside from the QPTAS of [14] for $\mathbb{R}^2$ and subsequent slight generalization of [1] no approximation scheme is known for CVRP on any non-trivial metrics for arbitrary values of Q (even for trees). Standard ways of extending a dynamic program for Euclidean metrics to bounded doubling metrics do not seem to work to extend the results of [14] to doubling metrics in quasi-polynomial time.

1.2 Overview of our technique We start by presenting a QPTAS for CVRP on trees and then extend the technique to graphs of bounded treewidth. Our main technique to design an approximation scheme for CVRP is to show the existence of a near-optimum solution where the sizes of the partial tours going past any node of the tree can be partitioned into only poly-logarithmic many classes. This will allow one to use dynamic programming to find a low-cost solution. A simple rounding of tour sizes to some threshold values (e.g. powers of $(1+\epsilon)$) only works (with some care) to achieve a bi-criteria approximation as any underestimation of tour sizes may result in tours that are violating the capacities. To achieve a true approximation (without capacity violation), we show how we can break the tours of an optimum solution into "top" and "bottom" parts at any node v (the bottom part of the tour being the part inside the subtree and then swap the bottom parts of tours with the bottom parts of other tours which are smaller, and then "round them up" to the nearest value from a set of poly-logarithmic threshold values. This swapping creates enough room to do the "round up" without violating the capacities. However, this will cause a small fraction of the vertices to become "not covered", we call them orphant nodes. We will show how we can randomly choose some tours of the optimum and add them back to the solution (at a small extra cost) and use these extra tours (after some modifications) to cover the orphant nodes. There are many details along the way. For instance, we treat the demand of each node as a token to be picked up by a tour. To ensure partial tour sizes are always from a small (i.e. poly-logarithmic) size set, we add extra tokens over the nodes. Also, for our QPTAS to work, we need to bound the height of the tree. We show how we can reduce the height of the tree to poly-logarithmic at a small loss using a height reduction lemma that might prove useful for other vehicle routing problems.

The technique of QPTAS for trees then can be extended to graphs of bounded treewidth and also graphs of bounded doubling dimension; prove the existence of a similar near-optimum solution and find one using the dynamic program. Or one can use the known results for the embedding of graphs of bounded doubling dimension into graphs of small treewidth. Many of the proofs are deferred to the full version.

2 Preliminaries

Recall that an instance $\mathcal{I}$ to CVRP is a graph $G = (V, E)$, where $w(e)$ is the cost or weight of edge $e \in E$ and Q is the capacity of the vehicle. Each tour $\mathcal{T}$ is a walk over some nodes of G. We say $\mathcal{T}$ "covers" node v if it serves the demand at node v. For the unit demand CVRP, it is easier to think of the demand of each node v as being a token on v that must be picked up by a tour. We can generalize this and assume each node v can have multiple tokens, and the total number of tokens a tour can pick is most Q (possibly from the same or different locations). Note that each tour might visit vertices without picking any token there. The goal is to find a collection of tours of minimum total cost such that each token is picked up (or, say, covered) by some tour. We use OPT(G) or simply OPT to refer to an optimum solution of G, and $\textsc{opt}$ to denote the value of it. Fix an optimal solution OPT. For any edge e let $f(e)$ denote the number of tours travelling edge e in OPT; so $\textsc{opt} = \sum_e w(e) \cdot f(e)$.

First we show the demand of each node is bounded by a function of Q. And then, using standard scaling and

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

rounding and at a small loss, we show we can assume the edge weights are polynomially bounded (in n). Given an instance for splittable CVRP with n nodes and capacity Q, it is possible that the demand $d(v) > Q$ for some node v. From the work of Adamaszek et al [1], we will show how we can assume that the demand at each node v satisfies $1 \leq d(v) < nQ$. Adamaszek et al [1] defined a *trivial* tour to be a tour that picks up tokens from a single node in T and a tour is *non-trivial* if the tour picks up tokens from at least two nodes in T. They defined a *cycle* to be a set of tours $t_1, \ldots, t_m (m \geq 2)$ and a set of nodes $\ell_1, \ell_2, \ldots, \ell_m, \ell_{m+1} = \ell_1$ such that each tour t_i covers locations ℓ_i and ℓ_{i+1} and the origin is not considered as a node in $\ell_1, \ldots, \ell_m$. They showed in Lemma 1 of [1] that there is an optimal solution in which there are no cycles. Since there are no 2-cycles, there are no two tours that cover the same pair of nodes. So there is an optimal solution such that there are at most n non-trivial tours (as argued in [1]). So putting aside trivial tours (each picking up Q tokens at a node), we can assume we have a total of at most nQ tokens, and in particular, each node has at most this many tokens. Without loss of generality, we assume we have removed as many trivial tours as we can so that each node has at most nQ demands (for a total of n^2Q demands). We can also assume there is at most one tour in OPT covering at most $Q/2$ demand. If there are at least two tours $\mathcal{T}_1$ and $\mathcal{T}_2$ covering less than $Q/2$ demand, they can be merged into a single tour at no additional cost. Since the total demand is at most n^2Q, the total number of tours in the optimal solution is at most $n^2Q/(Q/2) = 2n^2$.

Now we scale edge weights to be polynomially bounded. Observe that each tour in OPT traverses each edge e at most once in each direction, so at most twice. Suppose we have guessed the largest edge weight that belongs to OPT (by enumerating over all possible such guesses) and have removed any edge with weight larger. Let $W = \max_{e \in E} w(e)$ be the largest edge which clearly is used at least twice by OPT (since for each node v with zero demands in T_v we can delete the subtree T_v). Suppose we build instance $\mathcal{I}'$ by rounding up the weight of each edge e to be a maximum of $w(e)$ and $\epsilon W/4n^4$. Since there are a total of at most $2n^2$ tours in OPT and each edge is traversed at most twice by each tour, and there are at most n^2 edges, the cost of solution OPT in $\mathcal{I}'$ is at most $\text{OPT} + 4n^2 \cdot n^2 \cdot \frac{\epsilon W}{4n^4} \leq (1+\epsilon)\text{OPT}$. Note that the ratio of maximum to minimum edge weight in $\mathcal{I}'$ is $4n^4/\epsilon$, but the edge weights are not necessarily integer. Now suppose we scale the edge weights so that the minimum edge weight is 1 and the maximum edge weight is $4n^4/\epsilon$ and then scale them all by $1/\epsilon$, and then round each one up to the nearest integer. Note that by this rounding to the nearest integer, the cost of each edge is increased by a factor of at most $1+\epsilon$, so the cost of an optimum solution in the new instance is at most $(1+\epsilon)(1+\epsilon) = (1+O(\epsilon))$ factor larger than before rounding while the edge weights are all polynomially bounded integers. So from now on, we assume we have this property for the given instance at a small loss.

We will use the following two simplified versions of the Chernoff Bound [23] in our analysis.

LEMMA 2.1. **(Chernoff bound)** *Let $Y = \sum_{i=1}^{n} Y_i$ where $Y_i = 1$ with probability p_i and 0 with probability $1 - p_i$, and all Y_i's are independent. With $\mu = \mathbb{E}[Y]$, $\mathbb{P}[Y > 2\mu] \leq e^{-\mu/3}$ and $\mathbb{P}\big[Y < \frac{\mu}{2}\big] \leq e^{-\mu/8}$.*

3 QPTAS for CVRP on Trees

In this section we prove Theorem 1.1. We will first prove a structure theorem that describes the structural properties of a near-optimal solution. We will leverage these structural properties and use dynamic programming to compute a near-optimal solution.

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

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

THEOREM 3.1. *Given a tree T as an instance of CVRP and for any fixed $\epsilon > 0$, one can build a tree T' with height $\delta \log^2 n/\epsilon$, for some fixed $\delta > 0$, such that $\text{OPT}(T') \leq \text{OPT}(T) \leq (1+\epsilon)\text{OPT}(T')$. Furthermore, any feasible solution for T' can be transformed into a feasible solution of T while increasing the cost by at most a factor $(1+\epsilon)$.*

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

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

in approximation ratio.

3.1.1 Overview of the ideas Let us give a high-level idea of the Structure theorem. In order to do that, it is helpful to start from a simpler task of developing a bi-criteria approximation scheme[1]

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

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

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

Going from a Bicriteria to a true QPTAS: Our main tool to obtain a true approximation scheme

[1]Note that [9] already presents a bicriteria PTAS for CVRP on trees. We present a simple bi-criteria QPTAS here as it is our starting point towards a true approximation scheme.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

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

To achieve what we have outlined above, suppose T has height h (where $h = \delta \log^2 n/\epsilon$). Let V_ℓ (for $1 \le \ell \le h$) be the set of vertices at level ℓ of the tree where $V_1 = \{r\}$ and for each $\ell \ge 2$, V_ℓ are those vertices whose parent is in level $\ell - 1$. For every tour $\mathcal{T}$ and every level ℓ, the top part of $\mathcal{T}$ w.r.t. ℓ (denoted by $\mathcal{T}_\ell^{top}$), is the part of $\mathcal{T}$ induced by the vertices in $V_1 \cup \ldots \cup V_{\ell-1}$ and the bottom part of $\mathcal{T}$ are the partial tours of $\mathcal{T}$ in the subtrees rooted at a vertex in V_ℓ. Note that if we replace each partial tour of the bottom part of a tour $\mathcal{T}$ with a partial tour of a smaller capacity, the tour remains a capacity respecting tour. Consider a node v (which is at some level ℓ) and suppose we have n_v partial tours covering T_v. Let the n_v tours in increasing order of their coverage be $t_1, \ldots, t_{n_v}$. Let $|t_i|$ be the coverage of tour t_i (so $|t_i| \le |t_{i+1}|$). For a g (to be specified later), we add enough empty tours to the beginning of this list so that the number of tours is a multiple of g. Then, we will put these tours into groups $G_1^v, \ldots, G_g^v$ of equal sizes by placing the i'th n_v/g partial tours into G_i^v. Let $h_i^{v,max}$ ($h_i^{v,min}$) refer to the maximum (minimum) size of the tours in G_i^v. This grouping is similar to the grouping in the asymptotic PTAS for the classic bin-packing problem. Note that $h_i^{v,max} \le h_{i+1}^{v,min}$.

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

How to handle the case of orphant tokens (those picked by the tours in the last groups G_g^v before the swap)? We will show that if n_v is sufficiently large (at least polylogarithmic) then if we sample a small fraction of the tours of the optimum at random and add two copies of them (as extra tours), they can be used to cover the orphant tokens. So overall, we show how one can modify OPT by adding some extra tours to it at a cost of at most $\epsilon \cdot$ OPT such that: each node v has ≥ 1 tokens and the sketch of the partial tours at each node v is compact (only polylogarithmic many possible sizes) while the dropped tokens overall can be covered by the extra tours.

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

First, we assume that OPT has at least $d \log n$ many tours for some sufficiently large d. Otherwise, if there are at most $D = d \log n$ many tours in OPT we can do a simple DP to compute OPT: for each node v, we have

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

a sub problem $A[v, T_1^v, \ldots, T_D^v]$ which stores the minimum cost solution if T_i^v is the number of vertices, the i'th tour is covering in the subtree T_v. It is easy to fill this table in time $O(n^D)$ having computed the solutions for its children.

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

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

DEFINITION 2. *For a node v, the i-th bucket, b_i, contains the number of tours of OPT_ℓ having coverage between $[\sigma_i, \sigma_{i+1})$ tokens in T_v where σ_i is the i-th threshold value. We will denote a node and bucket by a pair (v, b_i). Let $n_{v,i}$ be the number of tours in bucket b_i of v.*

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

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

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

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

To handle (cover) orphant nodes, we are going to (randomly) select a subset of tours of OPT as "extra tours" and add them to OPT' and modify them such that they cover all the tokens that are now orphant (i.e. those that were covered by partial tours of $G_{i,g}^v$ for all big vertex/bucket pairs at level ℓ). Suppose we select each tour $\mathcal{T}$ of OPT with probability ϵ. We make two copies of the extra tour, and we designate both extra copies to one of the levels V_ℓ that it visits with equal probability. We call these the extra tours.

LEMMA 3.1. *The cost of extra tours selected is at most $4\epsilon \cdot \mathrm{OPT}$ w.h.p.*

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

Therefore, we can assume that the cost of all the extra tours added is at most $4\epsilon \cdot \text{OPT}$. Let X_ℓ be the set of extra tours designated to level ℓ. We assume we add X_ℓ when we are building OPT_ℓ (it is only for the sake of analysis). For each $v \in V_\ell$ and vertex/bucket pair (v, b_i), let $X_{v,i}$ be those in X_ℓ whose partial tour in T_v has a size in bucket b_i. Each extra tour in X_ℓ will not be picking any of the tokens in levels $V_{<\ell}$ (as they will be covered by the tours already in OPT_ℓ); they are used to cover the orphant tokens created by partial tours of $G^v_{i,g}$ for each big vertex/bucket pair (v, b_i) with $v \in V_\ell$; as described below.

LEMMA 3.2. *For each level V_ℓ, each vertex $v \in V_\ell$ and big vertex/bucket pair (v, b_i), w.h.p. $|X_{v,i}| \geq \frac{\epsilon^2}{\delta \log^2 n} \cdot n_{v,i}$.*

LEMMA 3.3. *Consider all $v \in V_\ell$, big vertex/bucket pairs (v, b_i) and partial tours in $G^v_{i,g}$. We can modify the tours in $X_{v,i}$ (without increasing the cost) and adding some extra tokens at v (if needed) so that:*

1. *The tokens picked up by partial tours in $G^v_{i,g}$ are covered by some tour in $X_{v,i}$, and*
2. *The new partial tours that pick up the orphant tokens in $G^v_{i,g}$ have size exactly $h^{v,max}_{i,g}$ and all tours still have size at most Q.*
3. *For each (new) partial tour of $X_{v,i}$ and every level $\ell' > \ell$, the size of partial tours of $X_{v,i}$ at a vertex at level ℓ' is also one of $O(\log Q \log^3 n/\epsilon^3)$ many sizes.*

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

We know $|G^v_{i,g}| = n_{v,i}/g = \frac{\epsilon^2}{2\delta \log n} \cdot n_{v,i}$. Using Lemma 3.2, we know with high probability that $|X_{v,i}| \geq \frac{\epsilon^2}{\delta \log^2 n} \cdot n_{v,i} = 2|G^v_{i,g}|$, so $|X_{v,i}|/|G^v_{i,g}| \geq 2$. Recall OPT' includes tours in OPT plus the extra tours in OPT that were sampled. Let $Y_{v,i}$ denote the number of tours in vertex/bucket pair (v, b_i) that were sampled, so $|X_{v,i}| = 2|Y_{v,i}|$ since we made two extra copies of each sampled tour and $|Y_{v,i}| \geq |G^v_{i,g}|$ with high probability. We will start by creating a one-to-one mapping $s : G^v_{i,g} \to Y_{v,i}$ which maps each tour in $G^v_{i,g}$ to a sampled tour in $Y_{v,i}$. We know such a one-to-one mapping exists since $|Y_{v,i}| \geq |G^v_{i,g}|$.

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

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

Also, note that for each partial tour $r_k \in G^v_{i,g}$ and for each level $\ell' > \ell$ if r_k visits a vertex $v' \in V_{\ell'}$, then the partial tour of r_k at $T_{v'}$ already satisfies the properties that: either its size belongs to a small vertex-bucket pair (v', b_i) (so has one of $O(\log^3 n/\epsilon)$ many possible values) or if it belongs to a big vertex bucket pair $(v', b_{i'})$ then its size is equal to $h^{v',max}_{i',j'}$ for some group j' and hence one of $O((\log Q \log n)/\epsilon^2)$ possible values. This implies that for the extra tours of $X_{v,i}$, after we reassign partial tours of $G^v_{i,g}$ to them (to cover the orphant

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

nodes), each will have a size exactly equal to $h_{i,g}^{v,max}$ at level ℓ and at lower levels $V_{>\ell}$ they already have one of the $O(\log Q \log^3 n/\epsilon^3)$ many possible sizes. This establishes the 3rd property of the lemma. $\square$

Therefore, using Lemma 3.3, all the tokens of T_v remain covered by partial tours; those partial tours in $G_{i,j}^v$ (for $1 \leq j < g$) are tied to the top parts of the tours from group $G_{i,j+1}^v$ and the partial tours of $G_{i,g}^v$ will be tied to extra tours designated to level ℓ. We also add extra tokens at v to be picked up by the partial tours of T_v so that each partial tour has size exactly equal to the maximum size of a group. All in all, the extra cost paid to build OPT_ℓ (from $\mathrm{OPT}_{\ell+1}$) is for the extra tours designated to level ℓ. The following theorem is an easy consequence of the previous lemmas.

THEOREM 3.2. ***(Structure Theorem)*** *Let* OPT *be the cost of the optimal solution to instance $\mathcal{I}$. We can build an instance $\mathcal{I}'$ on the same tree T such that each node has ≥ 1 tokens and there exists a near-optimal solution OPT' for $\mathcal{I}'$ having cost $(1+4\epsilon)$*OPT *w.h.p with the following property. The partial tours going down subtree T_v for every node v in OPT' has one of $O((\log Q \log^3 n)/\epsilon^3)$ possible sizes. More specifically, suppose (v, b_i) is a bucket pair for OPT'. Then either:*

- *b_i is a small bucket and hence there are at most $\alpha \log^3 n/\epsilon^2$ many partial tours of T_v whose size is in bucket b_i, or*
- *b_i is a big bucket; in this case there are $g = (2\delta \log n)/\epsilon^2$ many group sizes in b_i: $\sigma_i \leq h_{i,1}^{v,max} \leq \ldots \leq h_{i,g}^{v,max} < \sigma_{i+1}$ and every tour of bucket i has one of these sizes.*

3.2 Dynamic Program

In this section, we complete the proof of Theorem 1.1. We will describe how we can compute a solution of cost at most $(1+4\epsilon)$OPT using dynamic programming and based on the existence of a near-optimum solution guaranteed using the structure theorem. For each vertex/bucket pair, we do not know if the bucket is small or big, so we will consider subproblems corresponding to both possibilities. Informally, we will have a vector $\vec{n} \in [n]^\tau$ where if $i < 1/\epsilon$, n_i keeps track of the exact number of tours of size i and for $i \geq 1/\epsilon$, $\vec{n}_i$ keeps track of the number of tours in bucket b_i, or tours covering between $[\sigma_i, \sigma_{i+1})$ tokens. Let o_v denote the total number of tokens to be picked up across all nodes in the subtree T_v. Since each node has at least one token, $o_v \geq |V(T_v)|$. We will keep track of three other pieces of information conditioned on whether b_i is a small or big bucket. If b_i is a small bucket, we will store all the tour sizes exactly. Since the number of tours in a small bucket is at most $\gamma = \alpha \log^3 n/\epsilon^2$, we will use a vector $\vec{t^i} \in [n]^\gamma$ to represent the tours of a small bucket where $\vec{t^i_j}$ represents the size of j-th tour in bucket b_i. Suppose b_i is a big bucket, there are $g = (2\delta \log n)/\epsilon^2$ many tour sizes in the bucket corresponding to n^g possibilities. For each big bucket b_i at node v, we need to keep track of the following information,

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

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

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

Note that a bucket b_i is either small or big and cannot be both, hence given $(\vec{t}_v^i, \vec{h}_v^i, \vec{l}_v^i)$, it cannot be the case that $\vec{t}_v^i \neq \vec{0}, \vec{h}_v^i \neq \vec{0}$ and $\vec{l}_v^i \neq \vec{0}$. The subproblem $\mathbf{A}[v, \vec{y}]$ is supposed to be the minimum cost collection of partial tours going down T_v (to cover the tokens in T_v) and the cost of using the parent edge of v having a tour profile corresponding to $\vec{y}$. Our dynamic program heavily relies on the properties of the near-optimal solution in the structure theorem. Let v be a node. We will compute $A[\cdot,\cdot]$ in a bottom-up manner, computing $\mathbf{A}[v, \vec{y}_v]$ after we have computed the entries for the children of v.

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

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

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

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

Case 1: j = 1: This is the case when we restrict the coverage to only the first child of v, u_1.

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

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

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

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

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

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

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

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

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

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

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

DEFINITION 4. *We say configurations $\vec{y}'_v, \vec{y}'$ and $\vec{y}''$ are **consistent** if the following holds:*

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

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

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

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

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

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

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

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

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

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

computing $\mathbf{C}[o'_v, \vec{z}'_v, \vec{z}', \vec{z}'']$ takes $n^{O((\log Q \log^3 n)/\epsilon^3)}$ time. Since there are only $n^{O((\log Q \log^3 n)/\epsilon^3)}$ possibilities for $\vec{z}'_v, \vec{z}'$ and $\vec{z}''$, the cost of computing all entries of the consistency table is $n^{O((\log Q \log^3 n)/\epsilon^3)}$.

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

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

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

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

4 QPTAS for Bounded Treewidth Graphs

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

DEFINITION 5. *A **tree decomposition** of a graph G is a pair $(T, \{B_t\}_{t \in V(T)})$, where T is a tree whose every node $t \in V'$ is assigned a vertex subset $B_t \subseteq V(G)$, called a bag, such that the following three conditions hold:*

1. *$\cup_{t \in V(T)} B_t = V(G)$. In other words, every vertex of G is in at least one bag.*
2. *For every $uv \in E(G)$, there exists a node t of T such that bag B_t contains both u and v.*
3. *For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in B_t\}$, i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree of T.*

For a bag s, let C_s denote the union of nodes in bags below s including s. Bag s forms a boundary or border between nodes in C_s and $V(G) \setminus C_s$. We will assume an arbitrary bag containing the depot to be root of the tree decomposition. Let k be the treewidth of our graph G. We will assume that following properties hold for our tree decomposition T of G from the work of Boedlander and Hagerup [11],

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

To simplify notation, by replacing k' with k we will assume T has height $\delta \log n$ for some fixed $\delta > 0$ and each bag has width k. From the third property of a tree decomposition, we know that for every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree of

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

T. Since the bags associated with a node $u \in V(G)$ correspond to a subtree in T, we will place the demand/tokens of u at the root bag of the tree T_u i.e. the bag containing u closest to the root bag of T. Since T_u is a tree, we are guaranteed a unique root bag of T_u exists. We are doing this to ensure that the demand of a client is delivered exactly once.

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

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

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

The proof of the following structure theorem is similar to that of Theorem 3.2 (see full version):

THEOREM 4.1. ***(Structure Theorem)*** *Let* OPT *be the cost of the optimal solution to instance $\mathcal{I}$. We can build an instance $\mathcal{I}'$ such that each node has ≥ 1 tokens and there exists a near-optimal solution* OPT$'$ *for $\mathcal{I}'$ having expected cost $(1+2\epsilon)$*OPT *with the following property. The partial tours going down C_s for every bag s in* OPT$'$ *has one of $O((\log Q \log^2 n)/\epsilon^2)$ possible sizes. More specifically, suppose (s, b_i, x, z) is a entry/exit-bag-bucket configuration for* OPT$'$. *Then either:*

- *b_i is a small bucket and hence there are at most $\alpha \log^2 n/\epsilon$ many partial tours of C_s whose size is in bucket b_i, or*
- *b_i is a big bucket; in this case there are $g = (2\delta \log n)/\epsilon$ many group sizes in b_i: $\sigma_i \leq h_{i,1}^{s,x,z,max} \leq \ldots \leq h_{i,g}^{s,x,z,max} < \sigma_{i+1}$ and every tour of bucket i has one of these sizes.*

Using this structure theorem, we can design a dynamic program to find a near-optimum solution. The dynamic program will be similar to (but a lot more complex) the one developed for the case of trees (details in the full version). For a given bag s, we will estimate the number of tours entering and exiting s. Informally, we will have a vector $\vec{n}^{s,x,z} \in [n]^\tau$ where if $i < 1/\epsilon$, $\vec{n}_i^{s,x,z}$ keeps track of the exact number of tours covering i tokens in C_s by entering through x and exiting though z and if $i \geq 1/\epsilon$, $\vec{n}_i^{s,x,z}$ keeps track of the number of tours covering between $[\sigma_i, \sigma_{i+1})$ tokens. Let a_s denote the total number of tokens to be picked up from nodes from bags below and including bag s. Since each bag s has k nodes, we use $\vec{o}_s \in [n]^k$ to denote the extra tokens to be picked up from nodes at bag s. If v is a node in bag s, then $\vec{o}_{s,v}$ denotes the number of extra tokens to be picked up at v in s. For a given entry/exit-bag-bucket configuration (s, b_i, x, z), we will keep track of other pieces of information conditional on whether it is small or big. If entry/exit-bag-bucket configuration (s, b_i, x, z) is small, we will store all tour sizes exactly. Since the number of tours in a small entry/exit-bag-bucket configuration is at

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

most $\gamma = \alpha \log^2 n/\epsilon$, we will use a vector $\vec{t}^{s,x,z,i} \in [n]^\gamma$ to represent the tours where $\vec{t}_j^{s,x,z,i}$ represents the size of the j-th tour in the i-th bucket of tours covering C_s entering through x and exiting through z.

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

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

For a bag s and entry/exit pairs, let $\vec{p}_{s,x,z}$ be a vector containing information about all tours entering and exiting s through x and z across all buckets.

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

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

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

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

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

It can be shown that the time complexity of computing the DP table will be $(nk)^{O(k)} n^{O(k^2 \log Q \log^2 n/\epsilon^2)} = n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$. Hence, for the unit demand case, since $Q \leq n$, the runtime of our algorithm is $n^{O(k^2 \log^3 n/\epsilon^2)}$.

We can extend our algorithm for unit demand CVRP on bounded-treewidth graphs to the splittable CVRP when demands are quasi-polynomially bounded. In our algorithm for unit demand CVRP for bounded-treewidth CVRP, we viewed the unit demand of each node as a token placed at the node. For the splittable case, we can rescale the demand $d(v)$ such that there are $1 \leq d(v) < nQ$ tokens on a node and we can use the same structure theorem as before by modifying tours such that there are at most $O(\log Q \log^2 n/\epsilon^2)$ different tours for partial tours at a node. We can use the same DP to compute the solution. The time complexity of computing the entries of DP tables is, $(kQ)^{O(k)}(nQ)^{O(k^2 \log Q \log^2 n/\epsilon^2)} = (nQ)^{O(k^2 \log Q \log^2 n/\epsilon^2)}$ since $k \leq n$. Suppose $Q = n^{O(\log^c n)}$, then the runtime of our algorithm is $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$.

5 Extension to Splittable CVRP for Graphs of Bounded Doubling Metrics and Bounded Highway Dimension

In this section, we will show how we can use our algorithm for CVRP on bounded-treewidth graphs as a black box to obtain a QPTAS for graphs of bounded doubling metrics and graphs of bounded highway dimension. We will use the following result about embedding graphs of doubling dimension D into a bounded-treewidth graph of treewidth $k \leq 2^{O(D)} \left\lceil \left(\frac{4D \log \Delta}{\epsilon}\right)^D \right\rceil$ by Talwar [25].

Lemma 5.1. *(Theorem 9 in [25]) Let (X, d) be a metric with doubling dimension D and aspect ratio Δ. For any $\epsilon > 0$, (X, d) can be $(1 + \epsilon)$ probabilistically approximated by a family of treewidth k-metrics for $k \leq 2^{O(D)} \left\lceil \left(\frac{4D \log \Delta}{\epsilon}\right)^D \right\rceil$.*

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

We will also use the following result by Feldmann et al. [15] related to graphs of low highway dimension.

LEMMA 5.2. *(Theorem 3 in [15]) Let G be a graph with highway dimension D of violation $\lambda > 0$, and aspect ratio Δ. For any $\epsilon > 0$, there is a polynomial-time computable probabilistic embedding H of G with treewidth $(\log \Delta)^{O\left(\log^2(\frac{D}{\epsilon\lambda})/\lambda\right)}$ and expected distortion $1 + \epsilon$.*

For both graph classes, our algorithm works as follows. The input graph G is embedded into a host graph H of bounded treewidth using the embedding given in Lemma 5.1 and Lemma 5.2. The algorithm then finds a $(1+\epsilon)$-approximation for CVRP for H, using the dynamic programming solution from the Section 5. The solution for H is then *lifted* back to a solution in G. For each tour in the solution for H, a tour in G will visit nodes in the same order as the tour in H. The embedding given in Lemma 5.1 and Lemma 5.2 is such that an optimal set of tours in the host graph gives a $(1 + \epsilon)$ solution in G. The embedding also ensures that H has treewidth small enough that the algorithm runs in quasi-polynomial time.

THEOREM 5.1. *For any $\epsilon > 0$ and $D > 0$, there is a an algorithm that, given an instance of the splittable CVRP with capacity $Q = n^{\log^c n}$ and the graph has doubling dimension D with cost OPT, finds a $(1 + \epsilon)$-approximate solution in time $n^{O(D^D \log^{2c+D+3} n/\epsilon^{D+2})}$.*

Proof. This follows easily from Lemma 5.1 and using the algorithm for bounded-treewidth as a black box. In place of k, we will substitute $k = 2^{O(D)} \left\lceil \left(\frac{4D \log \Delta}{\epsilon}\right)^D \right\rceil$ into the runtime for the algorithm for bounded-treewidth which is $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$. Hence, we have an algorithm for graphs of bounded doubling dimension with runtime $n^{O(D^D \log^{2c+D+3} n/\epsilon^{D+2})}$. □

As an immediate corollary, since $\mathbb{R}^2$ has doubling dimension $\log_2 7 < 3$ [26], the above theorem implies an approximation scheme for unit demand CVRP on Euclidean metrics on $\mathbb{R}^2$ in time $n^{O(\log^6 n/\epsilon^5)}$ which improves on the run time of $n^{\log^{O(1/\epsilon)} n}$ of [14].

THEOREM 5.2. *For any $\epsilon > 0, \lambda > 0$ and $D > 0$, there is a an algorithm that, given an instance of the splittable CVRP with capacity $Q = n^{\log^c n}$ and a graph with highway dimension D and violation λ finds a $(1+\epsilon)$-approximate solution in time $n^{O(\log^{2c+3+\log^2(\frac{D}{\epsilon\lambda})\cdot\frac{1}{\lambda}} n/\epsilon^2)}$.*

Proof. This follows easily from Lemma 5.2 and using the algorithm for bounded-treewidth as a black box. In place of k, we will substitute $k = (\log \Delta)^{O\left(\log^2(\frac{D}{\epsilon\lambda})/\lambda\right)}$ into the runtime for the algorithm for bounded-treewidth which is $n^{O(k^2 \log^{2c+3} n/\epsilon^2)}$. Hence, we have an algorithm for graphs of bounded doubling dimension with runtime $n^{O(\log^{2c+3+\log^2(\frac{D}{\epsilon\lambda})\cdot\frac{1}{\lambda}} n/\epsilon^2)}$. □

6 Conclusion

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

Although our result implies a QPTAS with a better run time for CVRP on Euclidean plan $\mathbb{R}^2$ ($n^{O(\log^6 n/\epsilon^5)}$ vs the time of $n^{\log^{O(1/\epsilon)} n}$ of [14]), getting a PTAS remains an interesting open question. As discussed in [1], the difficult case appears to be when Q is polynomial in n (e.g. $Q = \sqrt{n}$). Another interesting question is to consider CVRP on planar graphs and develop approximation schemes for them, and more generally, graphs of bounded genus or minor free graphs.

References

[1] A. Adamaszek, A. Czumaj, and A. Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. In Y. Dong, D. Du, and O. H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 994–1003. Springer, 2009.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

[2] K. Altinkemer and B. Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6(4):149–158, 1987.

[3] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, Sept. 1998.

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

[5] A. Becker. A tight 4/3 approximation for capacitated vehicle routing in trees. In E. Blais, K. Jansen, J. D. P. Rolim, and D. Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[6] A. Becker, P. N. Klein, and D. Saulpic. A quasi-polynomial-time approximation scheme for vehicle routing on planar and bounded-genus graphs. In K. Pruhs and C. Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 12:1–12:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[7] A. Becker, P. N. Klein, and D. Saulpic. Polynomial-time approximation schemes for k-center, k-median, and capacitated vehicle routing in bounded highway dimension. In Y. Azar, H. Bast, and G. Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[8] A. Becker, P. N. Klein, and A. Schild. A PTAS for bounded-capacity vehicle routing in planar graphs. In Z. Friggstad, J. Sack, and M. R. Salavatipour, editors, *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 2019.

[9] A. Becker and A. Paul. A framework for vehicle routing approximation schemes in trees. In Z. Friggstad, J. Sack, and M. R. Salavatipour, editors, *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 112–125. Springer, 2019.

[10] J. Blauth, V. Traub, and J. Vygen. Improving the approximation ratio for capacitated vehicle routing. *CoRR*, abs/2011.05235, 2020.

[11] H. L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. In Z. Fülöp and F. Gécseg, editors, *Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995, Proceedings*, volume 944 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1995.

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

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

[14] A. Das and C. Mathieu. A quasipolynomial time approximation scheme for euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015.

[15] A. E. Feldmann, W. S. Fung, J. Könemann, and I. Post. A $(1+\epsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2015.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

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

[17] M. Haimovich and A. H. G. R. Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

[18] S.-y. Hamaguchi and N. Katoh. A capacitated vehicle routing problem on a tree. In K.-Y. Chwa and O. H. Ibarra, editors, *Algorithms and Computation*, pages 399–407, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[19] M. Khachay and R. Dubinin. PTAS for the euclidean capacitated vehicle routing problem in r^d. In Y. Kochetov, M. Khachay, V. L. Beresnev, E. A. Nurminski, and P. M. Pardalos, editors, *Discrete Optimization and Operations Research - 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19-23, 2016, Proceedings*, volume 9869 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2016.

[20] M. Khachay and Y. Ogorodnikov. QPTAS for the CVRP with a moderate number of routes in a metric space of any fixed doubling dimension. In I. S. Kotsireas and P. M. Pardalos, editors, *Learning and Intelligent Optimization - 14th International Conference, LION 14, Athens, Greece, May 24-28, 2020, Revised Selected Papers*, volume 12096 of *Lecture Notes in Computer Science*, pages 27–32. Springer, 2020.

[21] M. Khachay, Y. Ogorodnikov, and D. Khachay. An extension of the das and mathieu QPTAS to the case of polylog capacity constrained CVRP in metric spaces of a fixed doubling dimension. In A. V. Kononov, M. Khachay, V. A. Kalyagin, and P. M. Pardalos, editors, *Mathematical Optimization Theory and Operations Research - 19th International Conference, MOTOR 2020, Novosibirsk, Russia, July 6-10, 2020, Proceedings*, volume 12095 of *Lecture Notes in Computer Science*, pages 49–68. Springer, 2020.

[22] M. Labbé, G. Laporte, and H. Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991.

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

[24] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[25] K. Talwar. Bypassing the embedding: Algorithms for low dimensional metrics. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 281–290, New York, NY, USA, 2004. Association for Computing Machinery.

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

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited