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
MOHAMMAD R. SALAVATIPOUR[†], Department of Computing Science, University of Alberta

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 [14], 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 [15] 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 $n^{O(\log^6 n/\epsilon^5)}$.

CCS Concepts: • **Theory of computation → Routing and network design problems**.

Additional Key Words and Phrases: Approximation Scheme, Capacitated Vehicle Routing, Bounded Treewidth, Bounded Doubling Dimension

ACM Reference Format:
Aditya Jayaprakash and Mohammad R. Salavatipour. 2023. Approximation Schemes for Capacitated Vehicle Routing on Graphs of Bounded Treewidth, Bounded Doubling, or Highway Dimension. *ACM Trans. Algor.* 1, 1 (January 2023), 36 pages. https://doi.org/XXXXXXX.XXXXXXX

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 with some constraints on the vehicles (e.g. having limited capacity, or the distance a vehicle can travel in each

[*]A preliminary version of this paper has appeared in the proceedings of The 33rd ACM-SIAM Symp. on Disc. Algorithms (SODA) 2022.
[†]Supported by NSERC DG

Authors' addresses: Aditya Jayaprakash, jayaprak@ualberta.ca, Department of Computing Science, University of Alberta; Mohammad R. Salavatipour, mrs@ualberta.ca, Department of Computing Science, University of Alberta.

1549-6325/2023/1-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

tour). These problems have numerous applications in real world settings. The Capacitated Vehicle Routing Problem (CVRP) was introduced by Dantzig and Ramser in 1959 [14]. 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 [19]. Haimovich et al. [19] 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 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 [28], 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 [24], 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. *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 $n^{O(\log^4 n/\epsilon^3)}$. Our algorithm extends to the cases of splittable and unsplittable CVRP on trees when $Q = n^{O(\log^c n)}$, for some constant $c > 0$, and the algorithm runs in time $n^{O(\log^{2c+4} n/\epsilon^3)}$.*

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

Theorem 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 and unsplittable CVRP on graphs of bounded treewidth when $Q = n^{O(\log^c n)}$ for some constant $c > 0$, 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 3. *For any $\epsilon > 0$ and fixed $D > 0$, there is a an algorithm that, given an instance of the splittable CVRP or unsplittable CVRP with capacity $Q = n^{O(\log^c n)}$ (for a constant $c > 0$) 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 $n^{O(\log^{O(1/\epsilon)} n)}$ of QPTAS of [15]. We also show how our results in Theorem 2 implies the following.

THEOREM 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 or unsplittable CVRP with capacity $Q = n^{O(\log^c n)}$ (for some constant $c > 0$), finds a solution whose cost is at most $(1 + \epsilon)$ times the optimum in time $n^{O(\log^{2c+3+\log^2(\frac{D}{\epsilon\lambda})\cdot\frac{1}{\lambda}} n/\epsilon^2)}$.*

We shoude note that in the splittable and unsplittable cases we consider in these theorems, the demands $d(v)$ and Q are integers and Q is quasi-polynomially bounded (in n). However, for unsplittable CVRP when demands of the nodes and capacity Q are fractional numbers in $[0, 1]$ the problem is APX-hard. This follows easily from a simple reduction from Bin packing to unsplittable CVRP on a star with one leaf being the depot connected to the center with an edge of cost 1 and all the other leaves being the items in the bin-packing each having the demand being equal to the size of the item and connected to the center with a zero-cost edge. For such instances, deciding between whether the optimum is 2 or 3 is NP-hard. However, if all the items (and bin sizes) in bin packing are integers that are polynomially bounded (in n) then the problem has a PTAS. Very recently, for unsplittable CVRP on trees where $Q = 1$ and $d(v) \in [0, 1]$ [26] have presented a $(1.5 + \epsilon)$-approximation.

1.1 Related Works

CVRP generalizes the classic TSP problem (with $Q = n$). For general metrics, Haimovich et al. [19] 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 unsplittable CVRP, Friggstad et al. [17] presented better approximation algorithms including one with ratio 3.194.

For the case of trees, Labbé et al. [24] showed splittable CVRP is NP-hard and Golden et al. [18] showed unsplittable version is APX-hard and hard to approximate better than 1.5. For splittable CVRP (again on trees), Hamaguchi et al. [20] 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 [15] gave a quasi-polynomial-time approximation scheme (QPTAS) for CVRP in the Euclidean plane ($\mathbb{R}^2$). For the same metric and when Q is $O(\log n/\log\log n)$ or Q is $\Omega(n)$ was shown for the same metric by Asano et al. [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 [15], and using it as a subroutine. For high dimensional Euclidean spaces $\mathbb{R}^d$, Khachay et al. [21] showed a PTAS when Q is $O(\log^{1/d} n)$. For graphs of bounded doubling dimension, Khachay et al. [22] gave a QPTAS when the number of tours is polylog(n) and Khachay et al. [23] gave a QPTAS when Q is polylog(n).

The following results are all for when Q is some constant. 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. [21]. A PTAS for planar graphs was shown by Becker et al. [8] and a QPTAS for planar and bounded-genus graphs was shown by Becker et al. [6]. A PTAS for graphs of bounded highway dimension and an exact algorithm for graphs with bounded treewidth with running time $O(n^{\text{tw}Q})$ (where tw is the treewidth) 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 some constant.

So aside from the QPTAS of [15] 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 [15] to doubling metrics in quasi-polynomial time.

Very recently and after the first draft of this paper was made public, building upon ideas of [9] and this paper, Mathieu and Zhou [25] have presented a PTAS for splittable CVRP on trees. Their result builds upon and extends the ideas of [9] for decomposing a tree into "components" and showing existence of a near optimum solution where the number of tours covering the vertices in each component is $O_\epsilon(1)$, there are $O_\epsilon(1)$ levels of components, and the distances are of bounded aspect ratio. They also use the idea of changing tours as discussed in this paper (they refer to it as adaptive rounding). This allows one to compute such a near optimum one in polynomial time. Their result does not seem to extend to the more general classes of bounded treewidth or doubling dimensions without some new ideas. As mentioned earlier, for unsplittable CVRP on trees where $Q = 1$ and $d(v) \in [0, 1]$, Mathieu and Zhou [26] present a polynomial time $(1.5 + \epsilon)$-approximation.

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) 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 by proving the existence of a similar near optimum solution and finding one using dynamic program. Or one can use the known results for the embedding of graphs of bounded doubling dimension into graphs of small treewidth.

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. We use the terms "coverage" and "size" of a tour to refer to the number of demands a tour covers. 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. So the coverage/size of a tour is the number of tokens it picks. We can generalize this and assume each node v can have multiple tokens and the total number of tokens a tour can pick is at 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 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 cycles of size 2, 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 all trivial tours and so there is a total of at most nQ demands.

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. Also we can think of each tour as a walk. 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 a weight larger. Let $W = \max_{e \in E} w(e)$ be the largest (guessed) edge in OPT. Suppose we build instance $\mathcal{I}'$ by rounding up the weight of each edge e to be a maximum of $w(e)$ and $\epsilon W/2n^3$. Since there are a total of at most n 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 the solution OPT in $\mathcal{I}'$ is at most $\textsc{opt} + 2n \cdot n^2 \cdot \frac{\epsilon W}{2n^3} \leq (1+\epsilon)\textsc{opt}$. Note that the ratio of maximum to minimum edge weight in $\mathcal{I}'$ is $2n^3/\epsilon$, but the edge weights are not necessarily an integer. Now suppose we scale the edge weights so that the minimum edge weight is 1 and the maximum edge weight is $2n^3/\epsilon$ 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. To summarize: we assume our instance has been transformed (at a small loss at the cost of optimum) to a new instance satisfying the following properties:

- For each edge e: $w(e) \leq 2n^3/\epsilon^2$.
- The total number of tokens on the nodes is at most nQ.
- There are no trivial tours.
- There are at most n tours in an optimal solution.

We will use the following two simplified versions of the Chernoff Bound [27] in our analysis.

Lemma 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 $\mathbb{P}\left[Y < \frac{\mu}{2}\right] \leq e^{-\mu/8}$.*

3 QPTAS FOR CVRP ON TREES

In this section we prove Theorem 1. We assume the given tree T is rooted at the depot. 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 $(1+O(\epsilon))$opt) with certain properties which make it easy to find 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 $(1+O(\epsilon))$opt. Clearly the tours of OPT′ form a capacity respecting solution of $\mathcal{I}$ as well (of no more cost).

To be able to do that we need to be able to bound the height of the tree. A starting point in our structure theorem is to show that given input tree T, for any $\epsilon > 0$, we can build another tree $\tilde{T}$ of height $O(\log^2 n/\epsilon)$ such that the cost of an optimum solution in $\tilde{T}$ is within $1+\epsilon$ factor of the optimum solution to T. We can lift a near-optimum solution for $\tilde{T}$ into a near-optimum solution of T. Our instance $\mathcal{I}'$ is built on $\tilde{T}$. We will show the following in Subsection 3.6

Theorem 5. *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 $\text{opt}(T') \leq \text{opt}(T) \leq (1+\epsilon)\text{opt}(T')$.*

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) $1+\epsilon$ 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. Similar to the definition of coverage (or size) given for a tour in the previous section, the coverage (or size) 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 as a partial tour.

[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.

A Bicriteria QPTAS: First we try to explain the ideas of a bicriteria QPTAS, i.e. an algorithm in which the cost is within $(1+\epsilon)$ of optimum, but each tour may have size up to $(1+\epsilon)Q$ and the algorithm runs in quasi-polynomial time. For simplicity, assume T is binary (this is not crucial in the design of the DP). A subproblem would be defined based on each node $v \in T$ and the structure of partial tours going into T_v to pick up tokens in T_v at minimum cost. More specifically, 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 coverage for that portion is i), entry $\mathbf{A}[v, \vec{t}]$ in our table would store the minimum cost of covering T_v with (partial) tours whose coverage 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 sizes of the tours into $O(\log Q/\epsilon)$ buckets, where bucket i represents sizes 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 $|S| = O(\log Q/\epsilon) = O(\log n/\epsilon)$ since $Q = \text{poly}(n)$. 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 coverage 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.

This scheme requires a bit more care (and details) to get a bicriteria $(1+\epsilon)$-approximation as every time (in our DP recursion) we combine two partial tours with threshold size values to obtain a bigger partial tour, it may not have a threshold size value and we have to do a rounding. These roundings (at various levels) compound. Instead of using thresholds that are powers of $1+\epsilon$, we can define the thresholds based on powers of $1+\epsilon'$ where $\epsilon' = \frac{\epsilon^2}{\log^2 n}$: 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 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 coverage 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 coverage $t_i + t_j$ (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 sizes 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 $(1+O(\epsilon))Q$. This is why we get a bicriteria approximation. 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 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 polyogarithmic 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 polyogarithmic 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.

Suppose T has height h (where $h = \delta \log^2 n/\epsilon$). Let V_ℓ (for $1 \leq \ell \leq h$) be the set of vertices at level ℓ of the tree where $V_1 = \{r\}$ and for each $\ell \geq 2$, 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, 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 coverage, 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 $|t_i| \leq |t_{i+1}|$). 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 create linear grouping of these tours: we create groups $G_1^v, \ldots, G_g^v$ of equal sizes by placing the first n_v/g partial tours $t_1, \ldots, t_{n_v/g}$ into G_1^v, the 2nd n_v/g partial tours $t_{n_v/g+1}, \ldots, t_{2n_v/g}$ into G_2^v, and in general the i'th n_v/g partial tours in that list into group 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 $h_i^{v,max} \leq h_{i+1}^{v,min}$.

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 $h_{i-1}^{v,max} \leq h_i^{v,min} \leq |t|$. 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 because the top part of the tours whose bottom part has a partial tour in G_g^v are assigned to partial tours in G_{g-1}^v. So the partial tours in G_g^v don't have any top part (i.e. are not reached from the depot to v). we call the tokens picked by these partial tours *orphant* tokens. For now, assume that we add a few extra tours to OPT at a 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. More specifically, for each level ℓ suppose we add two extra copies of each tour of the optimum with probability ϵ/h to X_ℓ: X_ℓ will be the extra tours for vertices in V_ℓ that will be used to collect the orphan tokens from groups G_g^v for vertices $v \in V_\ell$. The cost of these extra tours added to cover the orphant tokens will be small. Also, if n_v is sufficiently big, then with high probability at least $|G_g^v|$ many tours will have a copy in X_ℓ. Those extra tours are enough to be able to pick up the orphan tokens (i.e. the top part of these extra tours can be used to reach the bottom parts of those partial tours and tokens that are orphant). So overall, we show how one can modify OPT by adding some extra tours to it at a cost of at most $\epsilon \cdot \textsc{opt}$ such that: each node v has ≥ 1 tokens and the sketch of the partial tours at each node v is compact (only polyogarithmic 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$_\ell$. Once this is done for all levels, the final solution will be OPT′. To obtain OPT$_\ell$ from OPT$_{\ell+1}$ we keep the partial tours at levels $\geq \ell$ the same as OPT$_{\ell+1}$ but we change the top parts of the tours and how the top parts are matched to the partial tours at level ℓ so that together they form capacity respecting solutions (tours of coverage at most Q) without increasing the cost of the solution too much (i.e. overall the new solution after going through all levels has cost at most $(1 + O(\epsilon))\textsc{opt}$). The goal of the transformation is to ensure that in the new (near optimum) solution we build, the partial tours going down each node v have a size that belongs to a set of size polylog(n), instead of poly(n).

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 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 $1 \leq i \leq \lceil 1/\epsilon \rceil$, 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 OPT$_{\ell+1}$ to obtain OPT$_\ell$.

Definition 2. *For a node v, the i-th bucket, b_i, contains the number of tours of* OPT$_\ell$ *having coverage belonging to $[\sigma_i, \sigma_{i+1})$ 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 $\alpha \geq \max\{1, 12\delta\}$.*

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 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 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$_\ell$ 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 $\mathcal{T} \in \mathbf{T}_\ell$, then for each such tour $\mathcal{T}$ if at level ℓ its partial tour was in a group $j < g$ of a big vertex bucket pair (v, i), that partial tour is replaced with a smaller partial tour from group $j-1$ of the same big vertex bucket pair; after adding extra tokens at v (if needed) the size of that partial tour is the maximum size from group $j-1$. All the 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 because the top part of their tour is mapped to the bottom part of tours from smaller groups. We describe how to add some tours from the depot to v so that they together with the partial tours that were in $G_{i,g}^v$ can be used to cover those orphant tokens. More speficially, the top parts of the tours whose bottom part belongs to $G_{i,g}^v$ are mapped to partial tours in other groups $G_{i,g'}^v$ with $g' < g$. For such tours, their bottom part still exists, but there is no top part, i.e. a tour that goes from the depot to v. We use some extra tours that come from the depot down to v and each will be matched with one partial tours in $G_{i,g}^v$ to make them a complete tour.

One important observation is that when we make these changes to make OPT$_\ell$ from OPT$_{\ell+1}$, for any partial tours at vertices at lower levels, ($V_{>\ell}$) their sizes remain the same. It is only the tour sizes going down a vertex at level ℓ that we are adjusting (by adding extra tokens). All the lower level partial tours remain unchanged (only their top parts may get swapped). So the sizes of partial tours at vertices in levels $> \ell$ do not change when we build OPT$_\ell$ from OPT$_{\ell+1}$. This property holds inductively as we go up the tree (from larger ℓ to smaller ℓ) and ensures that the lower level partial tours have one of polylogarithmic many sizes. More precisely, as we go up the levels to compute OPT$_\ell$, 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 $G^v_{i,g}$ for all big vertex bucket pairs at level ℓ).

Suppose we select each tour $\mathcal{T}$ of OPT with probability ϵ (note that OPT is the original solution before modifications). We duplicate each so we have two copies of each sampled tour and we designate both copies to one of the levels V_ℓ that it visits with equal probability. We call these the extra tours. The extra tours designated to a level ℓ are used to cover the orphant tokens when we build OPT_ℓ (from $\text{OPT}_{\ell+1}$). In fact, for each extra tour designated to level ℓ we only need the portion of that tour in the layers $V_1, \ldots, V_\ell$, i.e. we don't need (don't use) the part of those tours that go down below level ℓ. The algorithm to build each OPT_ℓ from $\text{OPT}_{\ell+1}$ is also illustrated in Algorithm 1

Algorithm 1. Converting OPT to a near optimum structured solution

```
1: For each tour $\mathcal{T}$ in OPT and each level $1 \le \ell < h$, add two copies of $\mathcal{T}$ to $X_\ell$ with probability
   $\epsilon/h$; (extra tours for level $\ell$)
2: Let $\ell = h$, $\text{OPT}_\ell = \text{OPT}$
3: For $\ell = h - 1$ downto 1 do
4:     Let $\text{OPT}_\ell = \text{OPT}_{\ell+1}$
5:     For each $v \in V_\ell$ and each bucket $b_i$ where $(v, b_i)$ is a big vertex bucket pair do
6:         Partition the partial tours of the $(v, b_i)$ into $g$ equal size groups (linear grouping):
           $G^v_{i,1}, \ldots, G^v_{i,g}$.
7:         Consider a 1-1 mapping $f$ from $G^v_{i,j}$ to $G^v_{i,j-1}$ for $j \ge 2$.
8:         For any two tours $t, t'$ visiting $T_v$ where partial tours of them belong to groups $G^v_{i,j}$ and
           $G^v_{i,j+1}$ $(j < g)$, respectively. and $f(t') = t$: change $t'$ by following the top part of $t'$ outside
           $T_v$ and then following the bottom part of $t$ at $v$ (bottom part of $t'$ is matched with another
           tour from $G^v_{i,j+2}$); add extra tokens at $v$ to be picked up by the new tour $t'$ so that its size
           becomes exactly $h^{v,max}_{i,j}$. (partial tours of $G^v_{i,g}$ now are not connected to the depot anymore;
           they will be fixed soon)
9:     Partial tours in $G^v_{i,g}$ (for each big vertex bucket pair $(v, b_i)$) will be assigned to extra tours in
       $X_\ell$ (see Lemma 4)
```

Lemma 2. *The cost of extra tours selected is at most* $4\epsilon \cdot \text{OPT}$ *w.h.p.*

Proof. Recall that $f(e)$ denotes the number of tours passing through e in OPT. The contribution of edge e to the optimal solution is $2 \cdot w(e) \cdot f(e)$ and we can write $\text{OPT} = \sum_{e \in E} 2 \cdot w(e) \cdot f(e)$. Let e be the parent edge of a node in $v \in V_\ell$. Suppose an extra tour is designated to level ℓ, we will only use it to cover orphant tokens from big buckets from nodes in V_ℓ. A node v would use an extra tour to cover orphant tokens only if one of v's buckets is a big bucket. From now on, we will assume the extra tours only pass through an edge e if $f(e) \ge \alpha \log^3 n/\epsilon^2$ (we can shortcut it otherwise).

For an edge e, let $f'(e)$ denote the number of sampled tours passing through e and since we use two copies of each sampled tour, $2f'(e)$ is the number of extra tours passing through e in OPT$'$. We can write $\text{OPT}' = \sum_{e \in E} 2 \cdot w(e) \cdot (f(e) + 2f'(e))$ and the cost of extra tours is $\sum_{e \in E} 2 \cdot w(e) \cdot 2f'(e)$. While modifying OPT to OPT$'$, each tour in the optimal solution is sampled with probability ϵ. Let e be an edge with $f(e)$ tours $\mathcal{T}_{e,1}, \ldots, \mathcal{T}_{e,f(e)}$ passing through it. Let $Y_{e,i}$ be a random variable which is 1 if tour $\mathcal{T}_{e,i}$ is sampled and 0 otherwise.

$$\mathbb{E}\left[Y_{e,i}\right] = \mathbb{P}\left[\mathcal{T}_{e,i} \text{ is sampled}\right] = \epsilon.$$

Let $f'(e) = Y_e = \sum_{i=1}^{f(e)} Y_{e,i}$. By linearity of expectations, we have

$$\mathbb{E}[f'(e)] = \mathbb{E}[Y_e] = \sum_{i=1}^{f(e)} \mathbb{E}\left[Y_{e,i}\right] = \sum_{i=1}^{f(e)} \epsilon = \epsilon \cdot f(e).$$

Our goal is to show $\mathbb{P}[Y_e > 2\mathbb{E}[Y_e]]$ is very low. Using Chernoff bound with $\mu = \mathbb{E}[Y_e] = \epsilon \cdot f(e) \geq \alpha \log^3 n/\epsilon \geq 6 \log n$.

$$\mathbb{P}[Y_e > 2\mathbb{E}[Y_e]] \leq e^{-(2\log n)} = \frac{1}{n^2}$$

The above concentration bound holds for a single edge e. Using the union bound, we can show this hold with high probability over all edges,

$$\sum_{e \in E} \mathbb{P}[Y_e > 2\mathbb{E}[Y_e]] \leq \frac{1}{n}.$$

We showed $f'(e) \leq 2\epsilon \cdot f(e)$ with high probability. Hence, with high probability, the cost of the extra tours is at most

$$\sum_{e \in E} 2 \cdot w(e) \cdot 2f'(e) \leq \sum_{e \in E} 2 \cdot w(e) \cdot 4\epsilon \cdot f(e) = 4\epsilon \sum_{e \in E} 2 \cdot w(e) \cdot f(e) = 4\epsilon \cdot \textsc{opt}.$$

■

Therefore, we can assume that the cost of all the extra tours added is at most $4\epsilon \cdot \textsc{opt}$. 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 been assigned to 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. *For each level V_ℓ, each vertex $v \in V_\ell$ and big vertex bucket pair (v, b_i), w.h.p. $|X_{v,i}| \geq \frac{\epsilon^2}{\delta \log^2 n} \cdot n_{v,i}$.*

Proof. Suppose (v, b_i) is a big vertex bucket pair at some level V_ℓ. Let $p_1, \ldots, p_{n_{v,i}}$ be the partial tours in vertex bucket pair (v, b_i). Let the tour in OPT corresponding to p_j be $\mathcal{T}$. Two copies of $\mathcal{T}$ are assigned to b_i if both of the following events are true:

- Let A_j be the event where tour $\mathcal{T}$ is sampled as an extra tour. Since each tour is sampled with probability ϵ, we have $\mathbb{P}\left[A_j\right] = \epsilon$.
- Let B_j be the event where tour $\mathcal{T}$ is assigned to level ℓ. There are $h = \delta \log^2 n/\epsilon$ many levels and since $\mathcal{T}$ (if sampled) is assigned to any one of its levels, $\mathbb{P}\left[B_j\right] \geq 1/h \geq \epsilon/(\delta \log^2 n)$.

Let Y_j be a random variable which is 1 if p_j is an extra tour in (v, b_i) and 0 otherwise.

$$\mathbb{E}\left[Y_j\right] = \mathbb{P}\left[Y_j = 1\right] = \mathbb{P}\left[A_j \wedge B_j\right] = \mathbb{P}\left[A_j\right] \cdot \mathbb{P}\left[B_j\right] \geq \epsilon^2/(\delta \log^2 n).$$

Let $Y_{v,i} = \sum_{j=1}^{n_{v,i}} Y_j$ be the random variable keeping track of the number of sampled tours in (v, b_i). The number of extra tours, $|X_{v,i}| = 2Y_{v,i}$ since we add two copies of a sampled tour to $X_{v,i}$. By linearity of expectation, we have

$$\mathbb{E}\left[|X_{v,i}|\right] = 2\mathbb{E}\left[Y_{v,i}\right] = 2\sum_{j=1}^{n_{v,i}} \mathbb{E}\left[Y_j\right] \geq \frac{2\epsilon^2}{\delta \log^2 n} \cdot n_{v,i}.$$

We want to show that $|X_{v,i}| \geq \frac{\mathbb{E}[|X_{v,i}|]}{2} \geq \frac{\epsilon^2}{\delta \log^2 n} \cdot n_{v,i}$ with high probability over all vertex bucket pairs.

Using Chernoff Bound with $\mu = \mathbb{E}\big[|X_{v,i}|\big] \geq \frac{2\epsilon^2}{\delta \log^2 n} \cdot n_{v,i} \geq 24 \log n$ since $n_{v,i} \geq \alpha \log^3 n/\epsilon^2$ and $\alpha \geq 12\delta$.

$$\mathbb{P}\left[|X_{v,i}| < \frac{\mathbb{E}\big[|X_{v,i}|\big]}{2}\right] \leq e^{-(3\log n)} = \frac{1}{n^3}$$

Note that the above equation only shows the concentration bound for a single vertex bucket pair. There are n nodes and each node has up to $\tau = \log n/\epsilon$ buckets, so the total number of vertex bucket pairs is at most $n \log n/\epsilon$. Suppose we do a union bound over all buckets, we get

$$\sum_{\text{all } (v,b_i) \text{ pairs}} \mathbb{P}\left[|X_{v,i}| < \frac{\mathbb{E}\big[|X_{v,i}|\big]}{2}\right] \leq \frac{1}{n}.$$

We showed that for each vertex bucket pair v, b_i, $|X_{v,i}| \geq \frac{\epsilon^2}{\delta \log^2 n} n_{v,i} \geq \alpha \log n/(2\delta)$ holds with high probability. ■

Lemma 4. *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, 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 $s : G^v_{i,g} \longrightarrow Y_{v,i}$ 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 bucket b_i. 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 nodes), each will have size exactly equal to $h^{v,max}_{i,g}$ 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. ■

Therefore, using Lemma 4, all the tokens of T_v remain covered by partial tours; those partial tours in $G^v_{i,j}$ (for $1 \leq j < g$) are tied to the top parts of the tours from group $G^v_{i,j+1}$ and the partial tours of $G^v_{i,g}$ 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 a size exactly equal to the maximum size of a group. All in all, the extra cost paid to build OPT_ℓ (from $\text{OPT}_{\ell+1}$) is for the extra tours designated to level ℓ.

One can easily verify that following holds for the near optimum solution OPT′ we build from OPT.

Observation 1. *If the original solution* OPT *was an unsplittable solution, i.e. the tokens $d(v)$ of each node was picked up by a single tour (where $d(v) \leq Q$), in the modified solution* OPT′, *for every vertex v, all the tokens at v (including the extra tokens added), are picked up by a single tour as well.*

This observation is used to show that if OPT is a solution for an unsplittable instance of CVRP then the near optimum solution OPT′ is also a feasible solution for unsplittable CVRP.

Theorem 6. ***(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: $\sigma_i \leq h^{v,max}_{i,1} \leq \ldots \leq h^{v,max}_{i,g} < \sigma_{i+1}$ and every tour of bucket i has one of these sizes.*

Proof. We will show how to modify OPT to a near-optimal solution OPT′. We start from $\ell = h$ and let $\text{OPT}_\ell = \text{OPT}$. For decreasing values of ℓ we show, for each ℓ, how to modify OPT_{l+1} to obtain OPT_ℓ. We do this in the following manner: we do not modify partial tours in small buckets.

However, for tours in big buckets, in each vertex bucket pair (v, b_i) in level $\ell - 1$, we place them into g 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. We have a mapping f from each partial tour in G_i^v to one in G_{i-1}^v for $i \in \{2, \ldots, g\}$. We modify OPT_ℓ to $\mathrm{OPT}_{\ell+1}$ 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). For each tour $t \in G_{i-1}^v$, we will add $h_{i-1}^{v,\max} - |t|$ many extra tokens at v. Note that by this change, the size of any tour such as $\mathcal{T}$ can only decrease and we are not violating feasibility of the tour because $h_{i-1}^{v,\max} \leq h_i^{v,\min}$. However, the tokens in T_v picked up by the partial tours in $G_{i,g}^v$ are not covered by any tours. We can use Lemma 4 to show how we can use extra tours to cover the partial tours in $G_{i,g}^v$ such that the new partial tours have size exactly $h_{i,g}^{v,\max}$.

We will inductively repeat this for levels $\ell - 2, \ell - 3, \ldots, 1$ and obtain $\mathrm{OPT}_1 = \mathrm{OPT}'$. Note that by adding extra tokens $h_{i-1}^{v,\max} - |t|$ for a tour $t \in G_{i-1}^v$, we are enforcing that the coverage of each tour is the maximum size of tours in its group. In a big bucket, there are $g = (2\delta \log n)/\epsilon^2$ many group sizes, so there are $O(\log n/\epsilon^2)$ possible sizes for tours in big buckets at a node. In a small bucket, there can be at most $\alpha \log^3 n/\epsilon^2$ many tours and since there are $\tau = O(\log Q/\epsilon)$ many buckets, there can be at most $O((\log Q \log^3 n)/\epsilon^3)$ many tour sizes covering T_v.

Using Lemma 2, we know the cost of the extra tours is at most $4\epsilon \cdot \text{OPT}$ with high probability, so the cost of $\text{OPT}' \leq (1 + 4\epsilon)\text{OPT}$. ■

3.2 Dynamic Program

In this section, we prove the first part of Theorem 1 (CVRP with unit demands on trees). We will describe how we can compute a solution of cost at most $(1 + 4\epsilon)\text{OPT}$ 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 (i.e. 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 $\vec{t}_j^i$ 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 lie 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.

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 $T_{u_1} \cup \ldots \cup T_{u_{j-1}}$ 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 three 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 $T_{u_1} \cup \ldots \cup T_{u_j}$.

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 they cannot both 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) $(\vec{t}^1, \vec{h}^1, \vec{l}^1), (\vec{t}^2, \vec{h}^2, \vec{l}^2), \dots, (\vec{t}^\tau, \vec{h}^\tau, \vec{l}^\tau)$ 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, 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 $Q \le n$, the runtime of our algorithm is $n^{O(\log^4 n/\epsilon^3)}$.

3.5 Extension to Splittable and Snsplittable CVRP

We can extend our algorithm for unit demand CVRP in trees and show how we can get a QPTAS for splittable and unsplittable CVRP as long as the demands are quasi-polynomially bounded. This will prove the 2nd part of Theorem 1. 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 $1 \le d(v) < nQ$ 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) $(\vec{t}^1, \vec{h}^1, \vec{l}^1), (\vec{t}^2, \vec{h}^2, \vec{l}^2), \dots, (\vec{t}^\tau, \vec{h}^\tau, \vec{l}^\tau)$ 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 $n^{O(\log^{2c+4} n/\epsilon^3)}$.

For unsplittable CVRP, first observe that $d(v) \le Q$ for each node v. Using Observation 1, whenever our algorithm serves a node v, it picks up all the tokens at that node completely. Therefore, the solution it generates serves each node by a single tour. In this case, the running time of the algorithm will be $n^{O(\log^{2c+3} n/\epsilon^3)}$.

3.6 Height reduction

In this section, we will prove Theorem 5. The first goal is to decompose the edge set of the tree T into edge-disjoint paths. We will do so using the following lemma, similar to Lemma 5 from Cygan et al. [13] to obtain such a decomposition in polynomial-time for a different problem.

LEMMA 5. *There exists a partitioning of the edge set of T into edge-disjoint paths $\mathcal{P}$, which can be grouped into $s = O(\log n)$ collections (called levels) $L_1, \dots, L_s$ such that the following hold: For every root-to-leaf path $P = e_1e_2, \dots, e_\ell$ in T (where e_1 is an edge incident to the root and e_ℓ is an edge incident to a leaf), e_1 belongs to a level 1 or level 2 path, and for each edge e_i $(i \geq 2)$ either e_i is part of the same path in $\mathcal{P}$ that contains e_{i-1}, or it is part of a path in $\mathcal{P}$ whose level index is one more than the level of the path containing e_{i-1}.*

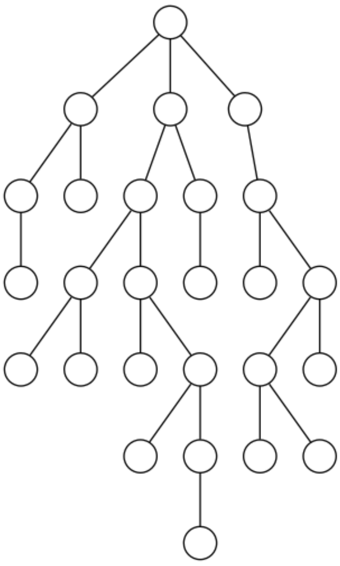
PROOF. Given a tree T, a D-path of T is a root-to-leaf path $P = v_1v_2 \dots v_k$ such that v_{i+1} is the child of v_i with the largest number of nodes in the tree rooted at $T_{v_{i+1}}$. If there are multiple children with the same number of descendants, break ties arbitrarily. Let P be a D-path. All the nodes in D-path P receive label 1. Also edges of P are level 1 edges. Let $T_1, \dots, T_c$ be the set of trees obtained from $T - P$. Let P_i be the D-path for T_i. We will label all nodes in P_i to be 2. Also all the edges in P_i, as well as the edges that connect the root of each T_i to P are level 2. We will repeat this process recursively by finding D-paths for trees resulting from $T_i - P_i$ and labelling every node in the D-path with the value corresponding to the depth of recursion (similarly for the edges of the D-paths we find as well as the edges that connect the current trees to the D-paths of the previous step). Each step involves finding a D-path, labelling the nodes and edges of the paths, deleting the paths and recursively repeating the process for the resulting trees (with the value of the label increased by 1). Nodes of D-paths of trees at depth ℓ in the recursion receive labels ℓ. We will terminate this process when all nodes have been labelled. Let L_j denote the collection of all D-paths whose nodes received the label j together with the edges that connects them to their parent (see figure 1).

Note that after the first step, the trees $T_1, \dots, T_c$ satisfy the property that $|V(T_i)| \leq \frac{|V(T)|}{2}$ i.e., each tree is at most half of the original tree. This is because we pick the child with the largest number of nodes in the subtree rooted at it. After each step, the size of the new components formed is at most half the size of the previous component, hence we would use at most $\log n$ labels to label all nodes in the tree. An easy induction shows that for each root-to-leaf path P the property stated in the lemma about the levels of edges of P hold since each root to leaf path either follows the same path as its parent or branches off into a new path with whose level is increased by 1. ■

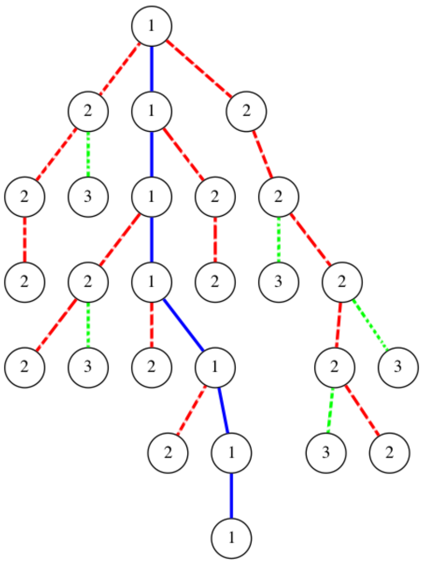
Figure 1 shows an example of such labelling where each color represents a level.

3.6.1 Creating a new tree. Given a tree T, we can use Lemma 5 to decompose the tree into edge-disjoint paths. Next, we describe an algorithm to modify the tree recursively into a low height tree $\tilde{T}$. The first step is to look at all the paths in L_1. L_1 is a special case since there is only one path in L_1 which goes from the depot to a leaf node. All the other levels L_i could have multiple disjoint paths. Let P be the path in L_1 and let $l(P)$ be the number of edges in path P. If $l(P) \leq \frac{\delta \log n}{\epsilon}$ for a $\delta > 0$ to be specified, then we are done for L_1.

However, if $l(P) > \frac{\delta \log n}{\epsilon}$, we will compress the path into a low height one. We will do a sequence of what is called up-pushes. We will pick $s \leq \frac{\delta \log l(P)}{\epsilon}$ points to be **anchor points**. Let us call the anchor points $a_1, \dots, a_s$ where a_1 is the anchor point closest to the root and a_s is closest to the leaf. We will later show how to find these anchor points.



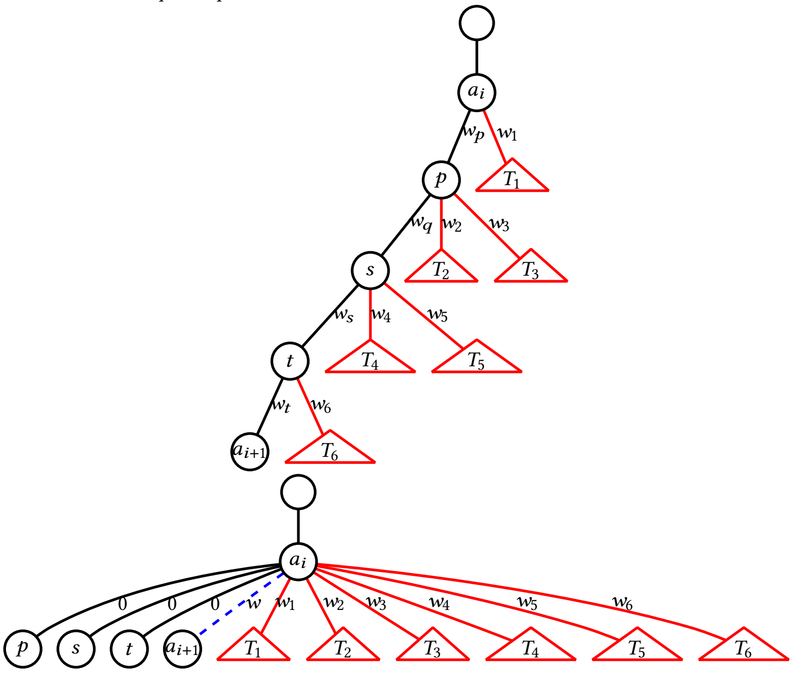
(a) A tree before labelling.



(b) Blue (solid) edges are level 1, red (dashed) edges are level 2 and green (dotted) edges are level 3.

Fig. 1. An example of a tree before and after applying labels to nodes

Fig. 2. A tree before an up-push (left) and after (right) with reduced height. The blue (dashed) edge connecting a_i and a_{i+1} has weight $w = w_p + w_q + w_s + w_t$



Each up-push acts on nodes in P between two consecutive anchor points a_i, a_{i+1} of the path P. During an up-push, we take all the nodes in P that lie between a_i and a_{i+1}, which we will call P', and make each node in P' a child of a_i with the edge connecting them to a_i having weight 0. Suppose there is a child subtree T_j, which is a child of a node in P' with edge connection cost w_j, the subtree T_j will become a child of a_i with the edge connecting them having cost w_j (see Figure 2). Once we have completed up-pushes for all paths in L_1, we will find anchor points and perform up-pushes for each path in L_2. We will repeat this for paths in L_i after our algorithm has finished up-pushes for paths in L_{i-1}.

We will now describe how we can find the anchor points. We will first describe what we would like to achieve from anchor points. We want the cost associated with a path in L_i for some tours to differ by at most $O(\epsilon)$ in our new tree compared to the original tree. Suppose P is a path in L_i and a tour t is travelling P down to node u which is between a_i and a_{i+1}. Then the cost of the portion of the tour from the root of P to a_i is the same in the original tree and the new tree; however the cost to travel from a_i to u is zero. We would like this cost in the original tree to be a small factor of the cost from the root of P to a_i.

Our algorithm to build $\tilde{T}$ from T works as follows from top to bottom. For any path P in L_i, we will set the top node of the path to be a_1 and its child in P to be a_2. Our goal is to pick a_i and a_{i+1} for $i > 2$ such that $w(a_i, a_{i+1}) > \epsilon \cdot w(a_1, a_i)$ and $w(a_i, v) \leq \epsilon \cdot w(a_1, a_i)$ where v is the last vertex on a_i, a_{i+1} path before a_{i+1}. If there is no a_{i+1} such that $w(a_i, a_{i+1}) > \epsilon \cdot w(a_1, a_i)$, then we set the last node of P to be a_{i+1}. So, we pick a_{i+1} to be the farthest vertex from a_i in P such that $w(a_i, v) \leq \epsilon \cdot w(a_1, a_i)$ where v is the last node before a_{i+1}. This in turn would imply that $w(a_1, a_{i+1}) > (1+\epsilon)w(a_1, a_i)$, except if a_{i+1} is the last node of the path. Hence, $w(a_1, a_i) > (1+\epsilon)^{i-2}w(a_1, a_2) > (1+\epsilon)^{i-2}$. Since edge weights are at most $2n^3/\epsilon^2$, the number of anchor points are at most $\frac{\delta \log n}{\epsilon}$ for some constant $\delta > 0$

3.6.2 Analysis. In the last section, we showed that every path in some level L_i can be made to have at most $O\left(\frac{\log n}{\epsilon}\right)$ nodes.

Lemma 6. *The height of the new tree $\tilde{T}$ is $O\left(\frac{\log^2 n}{\epsilon}\right)$.*

Proof. In our algorithm, we first decomposed T into a set of edge-disjoint paths. The decomposition guarantees that one would first visit a lower level node in any root-to-leaf path before visiting one with a higher level. Since there are at most $O(\log n)$ different levels, any root-to-leaf path will be a disjoint union of paths from levels $L_1, \ldots, L_s$ and there can be at most one path from each level. Since the height of a path in any level, L_i is at most $O\left(\frac{\log n}{\epsilon}\right)$, and there are at most $O(\log n)$ different levels, the maximum height in our new tree $\tilde{T}$ is at most $O\left(\frac{\log^2 n}{\epsilon}\right)$ ∎

Suppose we take a path P at some level L_c. Let us fix a tour in an optimal solution and let the farthest point in P the tour travels to be between anchor points $[a_i, a_{i+1})$. We use $[a_i, a_{i+1})$ denote that the tour crosses a_i but will not cross a_{i+1}. Let T be the original tree and let T' be the new tree with reduced height. A tour in the optimal solution for T' can visit nodes lying between a_i and a_{i+i} at no additional cost after visiting a_i. Suppose the cost of traversing the edges of P in T' is denoted by d, then the cost of traversing the edges of P in T is going to be at most $(1+O(\epsilon))d$. This is because the cost of the edges between a_i and the vertex before a_{i+1} sum to at most $O(\epsilon)w(r, a_i)$. Hence, the additional cost to cover them in T is only going to be at most an ϵ fraction more.

Lemma 7. *Let T be the original tree, $\tilde{T}$ be the new tree,* opt *be the cost of the optimal set of tours covering T and* opt′ *be the cost of the optimal set of tours covering $\tilde{T}$. Then,*

$$\textsc{opt}' \leq \textsc{opt} \leq (1+\epsilon)\textsc{opt}'.$$

Proof. Let us fix an optimal set of tours covering tree T with cost opt. Suppose we pick a tour t and decompose this tour into paths each of which is entirely within one level L_i. Suppose P is a path of t in some level L_c. Let the farthest point in P the tour travels to be between anchor points $[a_i, a_{i+1})$. In our construction, the cost to visit any point lying between the root of P and a_i is the same in both T and $\tilde{T}$. However, in $\tilde{T}$, the tour can visit any node lying between a_i and a_{i+1} for free, but the tour would have an additional cost to traverse these edges in tree T. Hence, for any path such P, the cost of a tour t to traverse edges in P is less in $\tilde{T}$ compared to T. Since any tour costs no more in instance $\tilde{T}$, we have opt′ ≤ opt.

Conversely, the extra cost of covering points lying between a_i and a_{i+1} in T is at most $O(\epsilon)$ times the cost of path P (based on the property of anchor points). This is because the cost of the edges between a_i and the vertex before a_{i+1} sum to at most $O(\epsilon)w(r, a_i)$. Hence, the additional cost to cover them in T is only going to be at most an ϵ fraction more. So the cost of using a path like P is at most an ϵ factor more in T compared to $\tilde{T}$. Thus, the cost of any tour t in T is at most $1+\epsilon$ times the cost of the same tour in $\tilde{T}$ and hence opt $\leq (1+\epsilon)$opt′ ■

Instead of T, we can solve the instance on $\tilde{T}$ with height $O(\log^2 n/\epsilon)$ and lift the solution for $\tilde{T}$ back to a solution for T. We obtain a solution for T with cost at most $(1+\epsilon)$opt.

4 QPTAS FOR BOUNDED TREEWIDTH GRAPHS

In this section we prove Theorem 2. First we start by recalling definitions of graphs of bounded treewidth.

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.

A graph $G = (V, E)$ has treewidth k if it has a tree decomposition in which each bag has size at most $k+1$. For such a graph 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**. 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 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 polyogarithmic 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 that the optimum solution has at least a polylogarithmic number of tours and that 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.

4.1 Changing OPT to a near-optimum structured solution

Similar to the structure theorem for trees, we will modify the optimal solution OPT to a near-optimum solution OPT′ having certain properties. We will start at the last level, and modify partial tours from OPT at level ℓ to obtain OPT_ℓ. We will then iteratively obtain $\mathrm{OPT}_{\ell-1}$ by modifying partial tours from OPT_ℓ at level $\ell - 1$, and iteratively do this for each level until we obtain $\mathrm{OPT}_1 = \mathrm{OPT}'$.

Definition 6. *For a bag s, the i-th bucket, b_i, entering at x and exiting at z contains the number of tours of* OPT_ℓ *having coverage between $[\sigma_i, \sigma_{i+1})$ tokens in C_s where σ_i is the i-th threshold value. We will denote this by a entry/exit-bag-bucket configuration (s, b_i, x, z). Let $n_{s,i}^{x,z}$ be the number of tours in bucket b_i entering through x and exiting through z in bag s.*

Definition 7. *An entry/exit-bag-bucket configuration (s, b_i, x, z) is **small** if $n_{s,i}^{x,z}$ is at most $\alpha \log^2 n/\epsilon$ and is **big** otherwise, for a constant $\alpha \geq \max\{1, 20\delta\}$.*

Note that for any bag s and entry/exit-bag-bucket configuration (s, b_i, x, z), if (s, b_i, x, z) is small, we do not modify the partial tours in it. However, if (s, b_i, x, z) is a big bucket, we create groups

$G^{s,x,z}_{i,1}, \ldots, G^{s,x,z}_{i,g}$ of equal sizes, for $g = (2\delta \log n)/\epsilon$; so $|G^{s,x,z}_{i,j}| = \lceil n^{x,z}_{s,i}/g \rceil$. We also consider a mapping f (as before) which maps (in the same order) the tours $t \in G^{s,x,z}_{i,j}$ to the tours in $G^{s,x,z}_{i,j-1}$ for all $1 < j \leq g$. Consider set $\mathbf{T}_\ell$ of all the tours $\mathcal{T}$ in OPT_ℓ that visit a bag in one of the lower levels $V_{\geq \ell}$. Consider an arbitrary such tour $\mathcal{T}$ that has a partial tour t in a big entry/exit-bag-bucket configuration (s, b_i, x, z), suppose t belongs to group $G^{s,x,z}_{i,j}$. We replace t with $f(t)$ in $\mathcal{T}$.

Now, add some extra tokens at x to be picked up by $\mathcal{T}$ so that the size of the partial tour of $\mathcal{T}$ at C_s is exactly $h^{s,x,z,\max}_{i,j-1}$. 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 entry/exit-bag-bucket configuration (s, b_i, x, z) is replaced with a smaller partial tour from group $j-1$ of the same big entry/exit-bag-bucket configuration; after adding extra tokens to x at bag s (if needed), the size is the maximum size from group $j-1$. The tokens that were picked by partial tours from $G^{s,x,z}_{i,g}$ for a big entry/exit-bag-bucket configuration (s, b_i, x, z) are now orphant. 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 $G^{s,x,z}_{i,g}$ for all big entry/exit-bag-bucket configuration (s, b_i, x, z) 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 bags at one of the levels V_ℓ that it visits with equal probability.

Lemma 8. *The expected cost of extra tours selected is* $2\epsilon \cdot \text{OPT}$.

Proof. Suppose $f^+(e)$ and $f^-(e)$ denote the number of tours traveling edge e in each of the two directions. So the contribution of edge e to the optimal solution is $2 \cdot w(e) \cdot (f^+(e) + f^-(e))$; $\text{OPT} = \sum_{e \in E} w(e) \cdot (f^+(e) + f^-(e))$. Let $m^+(e)$ ($m^-(e)$) denote the number of sampled tours from the tours contributing to $f^+(e)$ ($f^-(e)$). Since we used two copies for each sampled tour, the number of extra tours for an edge e is $2(m^+(e) + m^-(e))$. Let $\mathcal{T}_{e,1}, \ldots, \mathcal{T}_{e,f^+(e)+f^-(e)}$ be the tours using e in either directions. Like in the case of trees, it is possible for a tour to use edge e in both directions. Let $Y_{e,i}$ be a random variable which is 1 if tour $\mathcal{T}_{e,i}$ is sampled and 0 otherwise.

$$\mathbb{E}\big[Y_{e,i}\big] = \mathbb{P}\big[\mathcal{T}_{e,i} \text{ is sampled}\big] = \epsilon.$$

Let $m^+(e) + m^-(e) = Y_e = \sum_{i=1}^{f^+(e)+f^-(e)} Y_{e,i}$. By linearity of expectations, we have

$$\mathbb{E}\big[m^+(e) + m^-(e)\big] = \mathbb{E}[Y_e] = \sum_{i=1}^{f^+(e)+f^-(e)} \mathbb{E}\big[Y_{e,i}\big] = \sum_{i=1}^{f^+(e)+f^-(e)} \epsilon = \epsilon \cdot (f^+(e) + f^-(e)).$$

Summing up the extra cost over all edges, the expected cost of the extra tours is

$$2 \sum_{e \in E} \mathbb{E}\big[m^{in}(e) + m^{out}(e)\big] = 2\epsilon \cdot \sum_{e \in E} (f^+(e) + f^-(e)) = 2\epsilon \cdot \text{OPT}.$$

∎

Therefore, we can assume that the expected cost of all extra tours added is at most $2\epsilon \cdot \text{OPT}$. Let X_ℓ be the set of extra tours designated to bags in level ℓ. We assume we add X_ℓ when we are building OPT_ℓ (it is only for the sake of analysis). For each bag $s \in V_\ell$ and entry/exit-bag-bucket configuration (s, b_i, x, z), let $X^{s,x,z}_i$ be those in X_ℓ whose partial tour in C_s 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 $G^{s,x,z}_{i,g}$ for each big entry/exit-bag-bucket configuration (s, b_i, x, z) with $s \in V_\ell$; as described below.

Lemma 9. *For each level V_ℓ, each bag $s \in V_\ell$ and big entry/exit-bag-bucket configuration (s, b_i, x, z), w.h.p.* $|X^{s,x,z}_i| \geq \frac{\epsilon^2}{\delta \log n} \cdot n^{x,z}_{s,i}$.

Proof. Suppose (s, b_i, x, z) is a big entry/exit-bag-bucket configuration at some level V_ℓ. Let $p_1, \ldots, p_{n^{x,z}_{s,i}}$ be the partial tours in the entry/exit-bag-bucket configuration (s, b_i, x, z). Let the tour in OPT corresponding to p_j be $\mathcal{T}$. Two copies of tour $\mathcal{T}$ are assigned to b_i if both of the following events are true:

- Let A_j be the event where tour $\mathcal{T}$ is sampled as an extra tour. Since each tour is sampled with probability ϵ, we have $\mathbb{P}\left[A_j\right] = \epsilon$.
- Let B_j be the event where tour $\mathcal{T}$ is assigned to level ℓ. There are $h = \delta \log n$ many levels and since $\mathcal{T}$ (if sampled) is assigned to any one of its levels, $\mathbb{P}\left[B_j\right] \geq 1/h \geq 1/(\delta \log n)$.

Let Y_j be a random variable which is 1 if p_j is an extra tour in (v, b_i) and 0 otherwise.

$$\mathbb{E}\left[Y_j\right] = \mathbb{P}[Y_i = 1] = \mathbb{P}\left[A_j \wedge B_j\right] = \mathbb{P}\left[A_j\right] \cdot \mathbb{P}\left[B_j\right] \geq \epsilon/(\delta \log n).$$

Let $Y^{s,x,z}_i = \sum_{j=1}^{n^{x,z}_{s,i}} Y_j$ be the random variable keeping track of the number of sampled tours in (s, b_i, x, z). The number of extra tours, $|X^{s,x,z}_i| = 2Y^{s,x,z}_i$ since we add two copies of a sampled tour to $X^{s,x,z}_i$. By linearity of expectation, we have

$$\mathbb{E}\left[|X^{s,x,z}_i|\right] = 2\mathbb{E}\left[Y^{s,x,z}_i\right] = 2\sum_{j=1}^{n^{x,z}_{s,i}} \mathbb{E}\left[Y_j\right] \geq \frac{2\epsilon}{\delta \log n} \cdot n^{x,z}_{s,i}.$$

We want to show that $|X^{s,x,z}_i| \geq \frac{\mathbb{E}\left[|X^{s,x,z}_i|\right]}{2} \geq \frac{\epsilon}{\delta \log n} \cdot n^{x,z}_{s,i}$ with high probability over all vertex bucket pairs.

Using Chernoff Bound with $\mu = \mathbb{E}\left[|X^{s,x,z}_i|\right] \geq \frac{2\epsilon^2}{\delta \log^2 n} \cdot n^{x,z}_{s,i} \geq 24 \log n$ since $n^{x,z}_{s,i} \geq \alpha \log^2 n/\epsilon$ and $\alpha \geq 20\delta$.

$$\mathbb{P}\left[|X^{s,x,z}_i| < \frac{\mathbb{E}\left[|X^{s,x,z}_i|\right]}{2}\right] \leq e^{-(5 \log n)} = \frac{1}{n^5}$$

Note that the above equation only shows the concentration bound for a single entry/exit-bag-bucket configuration. For a bag, there are $O(k^2)$ many entry/exit pairs. There are $O(kn)$ bags and $\tau = O(\log n/\epsilon)$ buckets, so the total number of entry/exit-bag-bucket configuration is at most $O(k^2 n \log n/\epsilon)$. Suppose we do a union bound over all buckets, we get

$$\sum_{\text{all } (s,b_i,x,z) \text{ configurations}} \mathbb{P}\left[|X^{s,x,z}_i| < \frac{\mathbb{E}\left[|X^{s,x,z}_i|\right]}{2}\right] \leq \frac{1}{n}.$$

We showed that for every entry/exit-bag-bucket configuration (s, b_i, x, z), $|X^{s,x,z}_i| \geq \frac{\epsilon}{\delta \log n} n^{x,z}_{s,i}$ holds with high probability. ∎

Lemma 10. *Consider all bags $s \in V_\ell$, big entry/exit-bag-bucket configuration (s, b_i, x, z) and the partial tours in $G^{s,x,z}_{i,g}$. We can modify the tours in $X^{s,x,z}_i$ (without increasing the cost) and adding some extra tokens at nodes in s (if needed) so that:*

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

Proof. Our proof is going to be very similar to Lemma 4 for the case of trees. Our goal is to use the extra tours in $X^{s,x,z}_i$ to cover tokens picked up by partial tours of $G^{s,x,z}_{i,g}$ and we want each extra tour in $X_i^{s,x,z}$ to cover exactly $h_{i,g}^{s,x,z,\max}$ tokens. The tours in the last group, $G_{i,g}^{s,x,z}$, cover $\sum_{t\in G_{i,g}^{s,x,z}} |t|$ many tokens. We will add $\sum_{t\in G_{i,g}^{s,x,z}} (h_{i,g}^{s,x,z,\max} - |t|)$ extra tokens in node x at bag s for each entry/exit-bag-bucket configuration (s, b_i, x, z) so that there are $h_{i,g}^{s,x,z,\max}$ tokens corresponding to each partial tour in $G_{i,g}^{s,x,z}$. From now on, we will assume each partial tour in the last group $G_{i,g}^{s,x,z}$ covers $h_{i,g}^{s,x,z,\max}$ tokens.

Using Lemma 9, we know with high probability that $|X_i^{s,x,z}|/|G_{i,g}^{s,x,z}| \geq 2$ since $|X_i^{s,x,z}| \geq \frac{\epsilon}{\delta \log n} \cdot n_{s,i}^{x,z} = 2|G_{i,g}^{s,x,z}|$. Let $Y_i^{s,x,z}$ denote the number of tours in entry/exit-bag-bucket configuration (s, b_i, x, z) that were sampled, so $|X_i^{s,x,z}| = 2|Y_i^{s,x,z}|$ and $|Y_i^{s,x,z}| \geq |G_{i,g}^{s,x,z}|$ with high probability. We will start by creating a one-to-one mapping $s : G_{i,g}^{s,x,z} \to Y_i^{s,x,z}$ which maps each tour in $G_{i,g}^{s,x,z}$ to a sampled tour in $Y_i^{s,x,z}$. We know such a one-to-one mapping exists since $|Y_i^{s,x,z}| \geq |G_{i,g}^{s,x,z}|$.

Let $\mathcal{T}$ be a sampled tour in $Y_i^{s,x,z}$ with two extra copies of it, $\mathcal{T}_1$ and $\mathcal{T}_2$ in $X_i^{s,x,z}$. Let the partial tours of $\mathcal{T}$ at the bottom part in V_ℓ be $p_1, \ldots, p_m$. We know $|\mathcal{T}| \geq \sum_{i=1}^m |p_i|$. Like the case for trees, s maps at most one tour in $G_{i,g}^{s,x,z}$ to each p_j. If a tour from $G_{i,g}^{s,x,z}$ maps to p_j, we will assume the load assigned to p_j would be $r_j = h_{i,g}^{s,x,z,\max}$ and p_j has load 0 if no tour is assigned to it.

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 might not be able to fit all items $r_1, \ldots, r_m$ into a bin of size Q because $\sum_{i=1}^m |r_i| \leq (1+\epsilon)\sum_{i=1}^m |p_i| \leq (1+\epsilon)|\mathcal{T}| \leq (1+\epsilon)Q$. Similar to the case of trees, we can show that we can 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 $\mathcal{T}_2$ such that both $\mathcal{T}_1$ and $\mathcal{T}_2$ cover at most Q tokens and all items $r_1, \ldots, r_m$ are covered by either $\mathcal{T}_1$ or $\mathcal{T}_2$. Hence, we have shown that the extra partial tours pick up exactly $h_{i,g}^{s,x,z,\max}$ while picking up orphant tokens from $G_{i,g}^{s,x,z}$.

Also, the size of the extra tours after this modification at each bag s' at any level $\ell' > \ell$ is essentially the same as what each of r_i's were at those levels and since we go bottom to top in the tree, each of those partial tours r_i have a size that either belongs to a small bucket (and hence has one of $\alpha \log^2 n/\epsilon$ many sizes) or a big entry/exit-bag bucket (and hence has one of $O((\log Q \log n)/\epsilon^2)$ many sizes). Therefore, the size of partial tours of $X_i^{s,x,z}$ at any bag s' at level $\ell' > \ell$ is one of $O((\log Q \log^2 n)/\epsilon^2)$ many sizes. ∎

Therefore, using Lemma 10, all the tokens of C_s remain covered by partial tours; those partial tours in $G_{i,j}^{s,x,z}$ (for $1 \leq j < g$) are tied to the top parts of the tours from group $G_{i,j+1}^{s,x,z}$ and the partial tours of $G_{i,g}^{s,x,z}$ will be tied to extra tours designated to level ℓ. We also add extra tokens at nodes in s to be picked up by the partial tours of C_s 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 $\text{OPT}_{\ell+1}$) is for the extra tours designated to level ℓ.

Theorem 7. ***(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.*

Proof. We will show how to modify OPT to a near-optimal solution OPT′. We start from $\ell = h$ and let $\text{OPT}_\ell = \text{OPT}$. For decreasing values of ℓ we show, for each ℓ how to modify $\text{OPT}_{\ell+1}$ to obtain

OPT_ℓ. We do this in the following manner: we do not modify partial tours in small entry/exit-bag-bucket configuration. However, for tours in big entry/exit-bag-bucket configuration (s, b_i, x, z) in level $\ell - 1$, we place them into g groups $G^{s,x,z}_{i,1}, \ldots, G^{s,x,z}_{i,g}$ of equal sizes by placing the i'th $n^{x,z}_{s,i}/g$ partial tours into $G^{s,x,z}_{i,j}$. We have a mapping f from each partial tour in $G^{s,x,z}_{i,j-1}$ to one in $G^{s,x,z}_{i,j}$ for $j \in \{2, \ldots, g\}$. We modify OPT_ℓ to OPT_{l+1} in the following way: for each tour $\mathcal{T}$ that has a partial tour $t \in G^{s,x,z}_{i,j}$, replace the bottom part of $\mathcal{T}$ at s from t to $f(t)$ (which is in $G^{s,x,z}_{i,j-1}$). For each tour $t \in G^{s,x,z}_{i,j-1}$, we will add $h^{s,x,z,\max}_{i,j-1} - |t|$ many extra tokens at x in s. Note that by this change, the size of any tour such as $\mathcal{T}$ can only decrease and we are not violating feasibility of the tour because $h^{s,x,z,\max}_{i,j-1} \leq h^{s,x,z,\min}_{i,j}$. However, the tokens in C_s picked up by the partial tours in $G^{s,x,z}_{i,g}$ are not covered by any tours. We can use Lemma 10 to show how we can use extra tours to cover the partial tours in $G^{s,x,z}_{i,g}$ such that the new partial tours have size exactly $h^{s,x,z,\max}_{i,g}$.

We will inductively repeat this for levels $\ell - 2, \ell - 3, \ldots, 1$ and obtain $\mathrm{OPT}_1 = \mathrm{OPT}'$. Note that by adding extra tokens $h^{s,x,z,\max}_{i,j-1} - |t|$ for a tour $t \in G^{s,x,z}_{i,j-1}$, we are enforcing that the coverage of each tour is the maximum size of tours in its group. In a big bucket, there are $g = (2\delta \log n)/\epsilon$ many group sizes, so there are $O(\log n/\epsilon)$ possible sizes for tours in big entry/exit-bag-bucket configuration at a node. In a small entry/exit-bag-bucket configuration, there can be at most $\alpha \log^2 n/\epsilon$ many tours and since there are $\tau = O(\log Q/\epsilon)$ many buckets, there can be at most $O((\log Q \log^2 n)/\epsilon^2)$ many tour sizes covering C_b.

Using Lemma 8, we know the expected cost of the extra tours is at most $2\epsilon \cdot \textsc{opt}$, so the expected cost of $\textsc{opt}' \leq (1 + 2\epsilon)\textsc{opt}$. ■

4.2 Dynamic Program

In this section we prove Theorem 2 by presenting a dynamic program that will compute a near optimum solution guaranteed by the structure theorem (Theorem 7). 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 $i < 1/\epsilon$, $\vec{n}^{s,x,z}_i$ keeps track of the exact number of tours covering i tokens in C_s by entering through x and exiting though z and if $i \geq 1/\epsilon$, $\vec{n}^{s,x,z}_i$ 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 tokens (including 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 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 $\vec{t}^{s,x,z,i}_j$ 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 $\vec{h}^{s,x,z,i}_j = h^{s,x,z,\max}_{i,j}$, which is the size of the maximum tour which lies in group $G^{s,x,z}_{i,j}$ 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 $\vec{l}^{s,x,z,i}_j$ denotes the number of partial tours covering $h^{s,x,z,\max}_{i,j}$ tokens which lie in group $G^{s,x,z}_{i,j}$ 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 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 $v_1, \ldots, v_k$ 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 $\{v_1, \ldots, v_k\}$. Note that a tour can enter and exit s through the same node.

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

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.

For any nodes u, v in bag s, if there is no edge between u and v, we can add an edge between them and the cost of the edge is the shortest path cost between u and v in G. Similarly, for two adjacent bags, s and s_1, if $u \in s$ and $v \in s_1$ and if there is no edge between u and v in G, we will add an edge between them and the cost of the edge is the shortest path cost between u and v in G. If $u = v$, then the cost of the edge connecting them can be assumed to be zero. Let $\|\vec{o}_s\| = \sum_{u \in s} \vec{o}_{s,u}$.

For the base case, we consider leaf bags. A leaf bag s could have $a_s \geq 1$ tokens where $a_s = \|\vec{o}_s\|$. We will defer how we compute $\mathbf{A}[s, \vec{y}_s]$ to the end of this section. Informally, we will set $\mathbf{A}[s, \vec{y}_s]$ to be the minimum cost of the edges between nodes in bag s used for the tours in $\vec{y}_s$ to pick up $\vec{o}_s$ tokens located at nodes in bag s. The total coverage of the tours in $\vec{y}_s$ should be exactly a_s and a token at a node should be picked up by one of the tours in $\vec{y}_s$. From our structure theorem, we know there exists a near optimum solution such that each partial tour has one of $O(\log Q \log^2 n/\epsilon^2)$ tour sizes and for each small bucket, there are at most $\alpha \log^2 n/\epsilon$ partial tours in it. For every big bucket, there are $g = (2\delta \log n)/\epsilon$ many group sizes and every tour of bucket i has one of those sizes. We are computing all possible $\mathbf{A}[s, \vec{y}_s]$ entries and from our structure theorem, we know one of them has near-optimum expected cost, so by enumerating all possibilities, our dynamic program finds a near-optimums solution for the leaf bag, proving the base case.

Recall that the tree T is binary. Suppose bag s has two children in T, s_1 and s_2. To compute cell $\mathbf{A}[s, \vec{y}_s]$, we will use the entries of its children, $\mathbf{A}[s_1, \vec{y}']$ and $\mathbf{A}[s_2, \vec{y}'']$. Suppose C_{s_i} has a_{s_i} tokens, then $a_s = \|\vec{o}_s\| + a_{s_1} + a_{s_2}$. $\mathbf{H}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']$ checks whether the tour profiles $\vec{y}_s, \vec{y}'$ and $\vec{y}''$ are consistent meaning that all tokens picked up by tours in $\vec{y}'$ and $\vec{y}''$ along with tokens in s, $\vec{o}_s$ are picked up by tours in $\vec{y}_s$. We will also define $\mathbf{I}[\cdot,\cdot,\cdot,\cdot]$ where $\mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']$ denotes the cost of using the edges in bag s, edges connecting nodes in s and s_1, and edges connecting nodes in s and s_2. We can think of $\mathbf{I}$ as the cost of using edges to patch up partial tours covering C_{s_1} and partial tours covering C_{s_2} to create tours covering C_s. We will explain in the next section how $\mathbf{H}$ and $\mathbf{I}$ are computed. Recall $\vec{o}_s$ is part of $\vec{y}_s$. Suppose we have already computed the entries $\mathbf{A}[s_1, \cdot]$ and $\mathbf{A}[s_2, \cdot]$, we will compute $\mathbf{A}[s, \cdot]$ in the following way:

$$\mathbf{A}[s, \vec{y}_s] = \min_{\vec{y}', \vec{y}'' : \mathbf{H}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}''] = \text{True}} \{\mathbf{A}[s_1, \vec{y}'] + \mathbf{A}[s_2, \vec{y}''] + \mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']\}. \tag{1}$$

There are four possibilities for each partial tour t at bag s going down C_s covering tokens for the subtree rooted at children bags, s_1 and s_2 while also picking up extra tokens from nodes in s:

- t could be a tour that picks up tokens from nodes at bag s and does not visit or pick up tokens in $C_{s_1} \cup C_{s_2}$.
- t could be a tour that picks up tokens from nodes at bag s and picks up tokens only from C_{s_1}.
- t could be a tour that picks up tokens from nodes at bag s and picks up tokens only from C_{s_2}.
- t could be a tour that picks up tokens from nodes at bag s and picks up tokens from $C_{s_1} \cup C_{s_2}$.

We would find the minimum cost over all configurations $\vec{y}_s, \vec{y}', \vec{y}''$ as long as $\vec{y}_s, \vec{y}', \vec{y}''$ are consistent. We say $\vec{y}_s, \vec{y}', \vec{y}''$ are consistent if there is a way to write each tour in $\vec{y}_s$ as a combination of some tours from $\vec{y}'$ (at most $O(k^2)$ such tours), and some tours from $\vec{y}''$ (at most $O(k^2)$) while also picking up extra tokens from nodes in s. We would also require that all tokens in $\vec{y}'$ and $\vec{y}''$ are picked up by tours in $\vec{y}_s$.

For a leaf bag s, $\mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{0}, \vec{0}]$ denotes the minimum cost of tours entering bag s and visiting the nodes in s such that all tokens in s are picked up by some tour in $\vec{y}_s$. The last two entries are set to $\vec{0}$ since s is a leaf bag, and has no children, and there are no other tours (apart from those in $\vec{y}_s$) entering or exiting through nodes in bag s. We will set $\mathbf{A}[s, \vec{y}_s] = \mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{0}, \vec{0}]$ since $\mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{0}, \vec{0}]$ computes exactly the minimum cost collection of partial tours covering $C_s = s$ having tour profiles corresponding to $\vec{y}_s$. We will explain how to compute the entries of $\mathbf{I}[\cdot, \cdot, \cdot, \cdot]$ in the next section.

4.3 Checking Consistency

In our dynamic program, we are given three vectors $\vec{y}_s, \vec{y}', \vec{y}''$ where s is a bag having children bags s_1 and s_2. $\vec{y}'$ represents the configuration of tours covering C_{s_1} and $\vec{y}''$ represents the configuration of tours covering C_{s_2}. Given $\vec{y}_s$, for each node u in s, there are $\vec{o}_{s,u}$ many tokens to be picked up at u. We require the tokens for nodes in s and tokens covered by the partial tours from $\vec{y}'$ and $\vec{y}''$ to be picked up by tours in $\vec{y}_s$. For simplicity, we will refer to a tour from $\vec{y}_s$ as t_s, $\vec{y}'$ as t_u and a tour from $\vec{y}''$ as t_w.

Definition 8. *We say configurations $\vec{y}_s, \vec{y}'$ and $\vec{y}''$ are* ***consistent*** *if the following holds:*

- *Every tour in $\vec{y}'$ maps to some tour in $\vec{y}_s$.*
- *Every tour in $\vec{y}''$ maps to some tour in $\vec{y}_s$.*
- *Every tour in $\vec{y}_s$ has at most k^2 tours from $\vec{y}'$ and at most k^2 tours from $\vec{y}''$ mapping to it.*
- *Suppose $t^1_{s_1}, t^2_{s_1}, \ldots, t^{\sigma_1}_{s_1}$ are tours from $\vec{y}'$ and $t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ are tours from $\vec{y}''$ mapping to a tour t_s in $\vec{y}_s$. Then there is an ordering of $t^1_{s_1}, t^2_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ together with some edges in $s \cup s_1 \cup s_2$, which when concatenated gives t_s. Also, the number of extra tokens (from nodes in s) in total picked up by tour t_s from nodes in bag s is exactly $|t_s| - \sum_{j=1}^{\sigma_1} |t^j_{s_1}| - \sum_{j=1}^{\sigma_2} |t^j_{s_2}|$.*
- *All tokens of nodes at bag s, $\vec{o}_s$ are picked up tours in $\vec{y}_s$.*

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 from nodes in s. We will describe how we can compute consistency. Instead of using $\vec{y}_s$, we will use $\vec{z}_s$ which is the same as $\vec{y}_s$, but excludes information about the number of tokens in a bag, and only tracks information about the number of tours passing through bag s.

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

We will similarly define $\vec{z}'$ and $\vec{z}''$. Suppose t_{s,x_1,x_2} is a tour in s which enters through x_1 and exits through x_2, let $\vec{z}_s - t_{s,x_1,x_2}$ refers to the configuration $\vec{z}_s$ having one less tour of size $|t_{s,x_1,x_2}|$ from tours entering through x_1 and exiting through x_2 in s. Recall that $\vec{o}_s$ is the vector of extra tokens at each node in bag s which need to be covered by tours in $\vec{z}_s$.

Given $\vec{z}_s, \vec{z}', \vec{z}''$ and $\vec{o}_s$, we will use the table $\mathbf{H}$ to check if $\vec{z}_s, \vec{z}', \vec{z}''$ are consistent. We set $\mathbf{H}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ =True if $\vec{z}_s, \vec{z}'$ and $\vec{z}''$ are consistent and False otherwise. For the base case, $\mathbf{H}[\vec{0}, \vec{0}, \vec{0}, \vec{0}]$ =True. For the recurrence, we will look at all possible ways of combining tours from $\vec{z}'$ and $\vec{z}''$ into $\vec{z}_s$ while also picking up extra tokens from bag s. For a tour t_s, let $\vec{o}'_{s,t_s}$ be a vector where $\vec{o}'_{s,t_s,u}$ denotes the number of extra tokens picked up by t_s at node u in bag s. Let $\left\|\vec{o}'_{s,t_s}\right\| = \sum_{u \in s} \vec{o}'_{s,t_s,u}$ count the number of tokens picked up by t_s from nodes in s.

Note that the vertices in a bag s are a cut-set for the graph induced by the vertices in the subtree rooted at s (called T_s) and the rest of the graph. So any tour that visits a vertex that belongs to a bag in T_s must go through a vertex in s. Since each bag has k vertices, hence each tour t_s that enters and exits s at most k^2 times and can be can be obtained from concatenation of a collection of (at most k^2) tours $t^1_{s_1}, t^2_{s_1}, \ldots, t^{\sigma_1}_{s_1}$ entering and exiting s_1 from s, a colletion (of at most k^2) tours $t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ entering and exiting s_2 from s, and some edges P_{t_s} between vertices in $s \cup s_1 \cup s_2$, in some order. So similar to the case of trees, we can write the recurrence of our consistency table as:

$$\mathbf{H}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}''] = \bigvee_{\substack{t_s, t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}, \vec{o}'_{s,t_s} \\ |t_s| = \sum_{j=1}^{\sigma_1} |t^j_{s_1}| + \sum_{j=1}^{\sigma_2} |t^j_{s_2}| + \left\|\vec{o}'_{s,t_s}\right\|}} \mathbf{H}[\vec{o}_s - \vec{o}'_{s,t_s}, \vec{z}_s - t_s, \vec{z}' - t^1_{s_1} - \ldots - t^{\sigma_1}_{s_1}, \vec{z}'' - t^1_{s_2} - \ldots - t^{\sigma_2}_{s_2}], \tag{2}$$

where the $\bigvee$ is over all tours t_s, $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}$, $t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ where:

- $\sigma_1, \sigma_2 \le k^2$
- tour t_s is composed of concatenation of $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ plus some edges between vertices in $s \cup s_1 \cup s_2$, in some order.

Note that for each possible tour size in $\vec{z}_s$, we consider a tour t_s and an orderd collection of $O(k^2)$ tours from s_1, s_2 and edges from $s \cup s_1 \cup s_2$. The number of possible options for t_s is $O(k^2 \log Q \log^2 n/\epsilon^2)$, since each tour is defined by a pair of vertices of s (to enter and exit) and has one of $O(\log Q \log^2 n/\epsilon^2)$ possible sizes. Similar bound holds for each of $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$. Also, the number of ways tokens of t_s can be broken up among these subtours of s_1, s_2 is at most $Q^{O(k^2)}$. Therefore, for each entry of $\mathbf{H}[.,.,.]$ in Equation (2) the number of subproblems considered in $\bigvee$ is at most $(Qk)^{O(k^2)} \cdot \log^{O(k^2)} n/\epsilon^2$.

Although the above DP lets us check if $\vec{y}_s$, $\vec{y}'$ and $\vec{y}''$ are consistent, the entries of $\mathbf{H}$ are True/False and does not give us information about the cost associated with such composition of t_s from tours $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$. We will use table $\mathbf{I}[.,.,.]$ which is computed similar to table $\mathbf{H}[.,.,.]$ except that it stores the minimum cost of generating tours for s from patching up tours from s_1, s_2. Recall the recurrence of our dynamic program for $\mathbf{A}$ is the following,

$$\mathbf{A}[s, \vec{y}_s] = \min_{\vec{y}', \vec{y}'' : \mathbf{H}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}''] = \text{True}} \{\mathbf{A}[s_1, \vec{y}'] + \mathbf{A}[s_2, \vec{y}''] + \mathbf{I}[\vec{o}_s, \vec{y}_s, \vec{y}', \vec{y}'']\}.$$

The cost of using edges in C_{s_1} and C_{s_2} by the partial tours in $\vec{y}'$ and $\vec{y}''$ in $\vec{y}_s$ are accounted for by $\mathbf{A}[s_1, \vec{y}'] + \mathbf{A}[s_2, \vec{y}'']$. However, we have not accounted for the cost of hopping from one node to the other in s and also the cost of going from nodes in s to nodes in child bags, s_1 and s_2. Suppose t_s (a tour for s) is obtained from a collection of tours $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}$ from s_1 and tours $t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ from s_2 and a collection of edges P_{t_s} that are between the vertices in $s \cup s_1 \cup s_2$ in some order. We will let $\text{cost}(P_{t_s})$ denote the cost of the edges in P_{t_s}. The following figure illustrates an example of one such tour t_s (in red) and P_{t_s} (in dotted) and the tours from s_1 and s_2

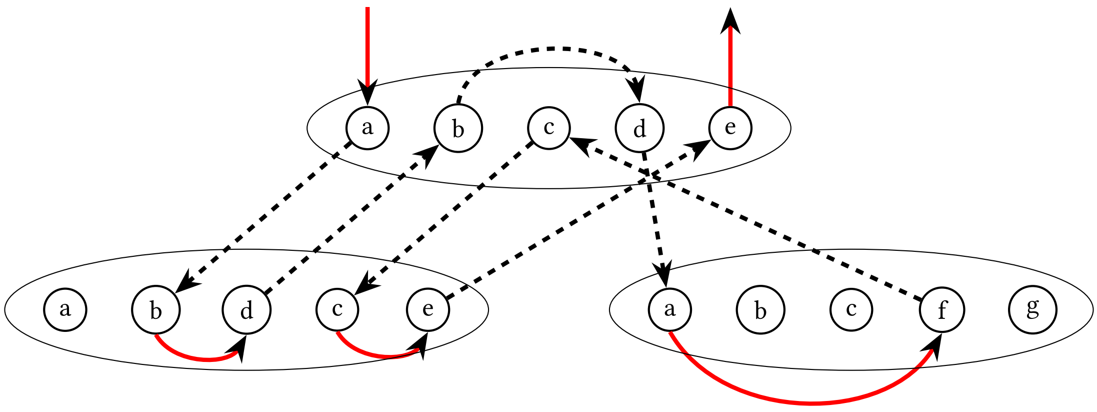


Fig. 3. Dashed edges represent one such edge set for a particular tour t_s

We will use $\mathbf{H}$ to compute $\mathbf{I}$. Let $\mathbf{I}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ denote the cost of using the edges in bag s, edges connecting nodes in s and s_1, and edges connecting nodes in s and s_2. We can think of $\mathbf{I}$ as the cost of using edges to patch up partial tours covering C_{s_1} ($\vec{z}'$), and partial tours covering C_{s_2} ($\vec{z}''$), to create tours covering C_s ($\vec{z}_s$). For the base case, we will set $\mathbf{I}[\vec{0}, \vec{0}, \vec{0}, \vec{0}] = 0$ and set all other entries to infinity. We will only compute an entry $\mathbf{I}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ if $\mathbf{H}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ =True. The computation for $\mathbf{I}$ is similar to $\mathbf{H}$ but also considers the cost of the edges from $s \cup s_1 \cup s_2$ used in concatenating the tours of s_1, s_2 to obtain a tour t_s for s. In our recurrence, we are taking a tour t_s from $\vec{y}_s$ along with tours $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}$ from $\vec{y}'$ (with $\sigma_1 \leq k^2$), tours $t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ from $\vec{y}''$ (with $\sigma_2 \leq k^2$), along with tokens $\vec{o}'_s$ that t_s covers at nodes in bag s. Suppose P_{t_s} is the ordered the set of edges from $s \cup s_1 \cup s_2$ that is used in between some ordering of $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ to obtain t_s. For such a tour t_s, there are $Q^{O(k^2)} k^{O(k^2)}$ many possibilities for P_{t_s} (their orders and the tokens from s picked by t_s). For a fixed P_{t_s}, $\mathrm{cost}(P_{t_s})$ is the cost edges of P_{t_s} used for forming t_s from concatenating tours of s_1 and s_2, while picking up extra tokens from nodes in s. We will enumerate through all possibilities, break the recurrence into subproblems and find a solution of minimum cost. We can write the recurrence as follows:

$$\mathbf{I}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}''] = \min \left\{ \mathrm{cost}(P_{t_s}) + \mathbf{I}[\vec{o}_s - \vec{o}'_{s,t_s}, \vec{z}_s - t_s, \vec{z}' - t^1_{s_1} - \ldots - t^{\sigma_1}_{s_1}, \vec{z}'' - t^1_{s_2} - \ldots - t^{\sigma_2}_{s_2}] \right\},$$

where the minimum is taken over all tours t_s, $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$, collection of edges P_{t_s} and $\vec{o}_{s,t_s}$ where:

- $|t_s| = \sum_{j=1}^{\sigma_1} |t^j_{s_1}| + \sum_{j=1}^{\sigma_2} |t^j_{s_2}| + \|\vec{o}'_{s,t_s}\|$
- $\sigma_1, \sigma_2 \leq k^2$
- tour t_s is composed of concatenation (in some order) of $t^1_{s_1}, \ldots, t^{\sigma_1}_{s_1}, t^1_{s_2}, \ldots, t^{\sigma_2}_{s_2}$ and edges P_{t_s}.

From the definition of $\mathbf{I}$, $\mathbf{H}$, and $\mathbf{A}$, it should be easy to see the correctness of recurrence given in Equation (1).

4.4 Time Complexity

We will work bottom-up and analyze the time complexity of $\mathbf{A}[\cdot, \cdot]$ on the assumption that we have already precomputed our consistency table $\mathbf{I}[\cdot, \cdot, \cdot, \cdot]$. Computing $\mathbf{A}[s, \cdot]$ requires looking at entries of child bags in $\mathbf{A}[\cdot, \cdot]$. Given $\vec{y}_s, \vec{y}'$ and $\vec{y}''$ which are consistent, computing the cost of $\mathbf{A}[s, \vec{y}_s]$ takes $O(1)$ time. Each $\vec{y}_s$ consists of $O(k^2)$ different $\vec{p}_{s,u,v}$ vectors. Each $\vec{p}_{s,u,v}$ contains τ many triples $(\vec{t}^{s,x,z,i}, \vec{h}^{s,x,z,i}, \vec{l}^{s,x,z,i})$.

(1) Each $\vec{t}^{s,x,z,i}$ has $n^{O(\log^2 n/\epsilon)}$ possibilities since there are at most $O(\log^2 n/\epsilon)$ tours in a small bucket.
(2) Each $\vec{h}^{s,x,z,i}$ and $\vec{l}^{s,x,z,i}$ have $n^{O(g)}$ possibilities. Recall that $g = (2\delta \log n)/\epsilon$, so each $\vec{h}^{s,x,z,i}$ and $\vec{l}^{s,x,z,i}$ have $n^{O(\log n/\epsilon)}$ possibilities.
(3) Each triple $(\vec{t}^{s,x,z,i}, \vec{h}^{s,x,z,i}, \vec{l}^{s,x,z,i})$ has $n^{O(\log^2 n/\epsilon)}$ possibilities.
(4) Since $\vec{p}_{s,u,v}$ has $\tau = O(\log Q/\epsilon)$ many such triples, the number of possible entries for $\vec{p}_{s,u,v}$ is $n^{O(\tau \log^2 n/\epsilon)} = n^{O(\log Q \log^2 n/\epsilon^2)}$.
(5) Since $\vec{y}_s$ consists of $O(k^2)$ different entries of $\vec{p}$, so the total number of possible entries for each $\vec{y}_s$ is $n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$.

Since there are $n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$ possibilities for $\vec{y}_s, \vec{y}'$ and $\vec{y}''$, the time of computing DP entries of $\mathbf{A}[s, \cdot]$ for a single bag s would take $n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$ and across all bags of the tree decomposition, it would still be $n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$.

Now, we will analyze the time of computing the consistency table $\mathbf{I}[\cdot,\cdot,\cdot,\cdot]$. Assuming we have computed smaller entries, the cost of computing if $\mathbf{I}[\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}'']$ requires taking all possibilities way of picking t_s, $t^1_{s_1}, \dots, t^{\sigma_1}_{s_1}$, $t^1_{s_2}, \dots, t^{\sigma_2}_{s_2}$, and P_{t_s} and some ordering of them, and $\vec{o}'_{s,t_s}$. Since there are at most $O(\log Q \log^2 n/\epsilon^2)$ different tour sizes, the number of possible ways of picking $t_s, t^1_{s_1}, \dots, t^{\sigma}_{s_1}, t^1_{s_2}, \dots, t^{\sigma_2}_{s_2}$ is $O((\log Q \log^2 n/\epsilon^2)^{3k^2})$. Since the number of entries in the vector of $\vec{o}$ is $O(k)$, there are $Q^{O(k)}$ possibilities for $\vec{o}'_s$. Each path P_{t_s} consists of $O(k)$ nodes and at most Q tokens can be picked up from each node, this would lead to $O(Q^k k^{k^2}) = (nk)^{O(k^2)}$ many possibilities for P_{t_s} since $Q \le n$. Hence, the total cost of computing a single entry of $\mathbf{I}[\cdot,\cdot,\cdot,\cdot]$ is $(nk)^{O(k^2)}$. Similar to the analysis for $\mathbf{A}[\cdot,\cdot]$, there are $n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$ possibilities for $\vec{o}_s, \vec{z}_s, \vec{z}', \vec{z}''$, hence the total cost of computing $\mathbf{I}[\cdot,\cdot,\cdot,\cdot]$ is $(nk)^{O(k^2)} n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$. Similarly, the cost of computing $\mathbf{H}[\cdot,\cdot,\cdot,\cdot]$ is $(nk)^{O(k^2)} n^{O(k^2 \log Q \log^2 n/\epsilon^2)}$.

Since the cost of computing $\mathbf{I}[\cdot,\cdot,\cdot,\cdot]$ dominates the cost of computing $\mathbf{A}[\cdot,\cdot]$, the total time complexity of our algorithm is $(nk)^{O(k^2 \log Q \log^2 n/\epsilon^2)}$. Hence, for the unit demand case, since $Q \le n$, the run time of our algorithm is $(nk)^{O(k^2 \log^3 n/\epsilon^2)}$.

4.5 Extension to Splittable and Unsplittable CVRP in Bounded Treewidth Graphs

We will 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 $1 \le d(v) < nQ$ 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. Each $\vec{y}_s$ consists of $O(k^2)$ different $\vec{p}_{s,u,v}$ vectors. Each $\vec{p}_{s,u,v}$ contains τ many triples $(\vec{t}^{s,x,z,i}, \vec{h}^{s,x,z,i}, \vec{l}^{s,x,z,i})$.

(1) Each $\vec{t}^{s,x,z,i}$ has $(nQ)^{O(\log^2 n/\epsilon^2)}$ possibilities since there are at most $O(\log^2 n/\epsilon)$ tours in a small bucket.
(2) Each $\vec{h}^{s,x,z,i}$ and $\vec{l}^{s,x,z,i}$ have $(nQ)^{O(g)}$ possibilities. Recall that $g = (2\delta \log n)/\epsilon^2$, so each $\vec{h}^{s,x,z,i}$ and $\vec{l}^{s,x,z,i}$ have $(nQ)^{O(\log n/\epsilon^2)}$ possibilities.
(3) Each triple $(\vec{t}^{s,x,z,i}, \vec{h}^{s,x,z,i}, \vec{l}^{s,x,z,i})$ has $(nQ)^{O(\log^2 n/\epsilon)}$ possibilities.
(4) Since $\vec{p}_{s,u,v}$ has $\tau = O(\log Q/\epsilon)$ many such triples, the number of possible entries for $\vec{p}_{s,u,v}$ is $(nQ)^{O(\tau \log^2 n/\epsilon)} = (nQ)^{O(\log Q \log^2 n/\epsilon^2)}$.

(5) Since $\vec{y}_s$ consists of $O(k^2)$ different entries of $\vec{p}$, the total number of possible entries for each $\vec{y}_s$ is $(nQ)^{O(k^2 \log Q \log^2 n/\epsilon^2)}$.

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

For unsplittable CVRP, similar to the case of trees, observe that $d(v) \leq Q$ for each node v and whenever our algorithm serves a node v, it picks up all the tokens at that node completely. Therefore, the solution it generates serves each node in a single tour.

5 EXTENSION TO 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 blackbox to obtain a QPTAS for graphs of bounded doubling metrics and graphs of bounded highway dimension. Consider a metric (V, d) defined on a set of vertices V along with distance function d between vertices. Let $B(v, r)$ be the ball of radius r around v, i.e., $B(v, r) = \{u : d(v, u) \leq r\}$. The doubling dimension of (V, d) is the smallest κ such that any $B(v, 2r)$ is contained in the union of at most 2^κ balls of radius r. A metric is called a doubling metric if κ is a constant. For example, a constant dimensional Euclidean metric is a doubling metric.

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 [29].

Lemma 11. *(Theorem 9 in [29]) 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 $k \leq 2^{O(D)} \left\lceil \left(\frac{4D \log \Delta}{\epsilon}\right)^D \right\rceil$.*

For graphs of bounded highway dimension we use the following definition from [16]. Suppose (V, d) is a metric space. We can think of V as the vertices of a complete graph G and d as the edge weights satisfying triangle inequality.

Definition 9. *[16] The highway dimension of (V, d) is the smallest integer κ such that, for some universal constant $c \geq 4$, for every $r \in \mathbb{R}^+$ and every ball $B(v, c \cdot r)$ of radius $c \cdot r$, there are at most κ vertices in $B(v, c \cdot r)$ hitting all shortest paths of length more than r that lie in $B(v, c \cdot r)$.*

The parameter $\lambda = c - 4$ is called the violation parameter. The following result by Feldmann et al. [16] shows how every graph with low highway dimension can be embedded approximately into a graph with (relatively) small treewidth.

Lemma 12. *(Theorem 3 in [16]) 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 11 and Lemma 12. 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 11 and Lemma 12 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 8. *For any $\epsilon > 0$ and $D > 0$, there is a an algorithm that, given an instance of the splittable or unsplittable 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 11 and using the algorithm for bounded-treewidth as a blackbox. 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 $\log_2 7 < 3$ [30], 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 [15].

Theorem 9. *For any $\epsilon > 0, \lambda > 0$ and $D > 0$, there is a an algorithm that, given an instance of the splittable or unsplittable 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 12 and using the algorithm for bounded-treewidth as a blackbox. 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 treewidths, 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. As we said earlier, following the anouncement of this paper, Mathieu and Zhou [25] presented a PTAS for CVRP on trees but the problem remains open to obtain a PTAS for the more general classes. For unsplittable CVRP, our result shows the difficult case is when demands (and Q) are not bounded, i.e. when $Q = 1$ and $d(v) \in [0,1]$. It would be interesting to see if there is an asymptotic PTAS for special graph classes (such as trees).

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 [15]), 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.

Acknowledgements: We thank the anonymous referres whose comments helped improve the writing of this paper.

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.

[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] M. Cygan, F. Grandoni, S. Leonardi, M. Pilipczuk, and P. Sankowski. A path-decomposition theorem with applications to pricing and covering on trees. In L. Epstein and P. Ferragina, editors, *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7501 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2012.

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

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

[16] 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.

[17] Z. Friggstad, R. Mousavi, M. Rahgoshay, and M. R. Salavatipour. Improved approximations for capacitated vehicle routing with unsplittable client demands. In K. Aardal and L. Sanità, editors, *Integer Programming and Combinatorial Optimization - 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27-29, 2022, Proceedings*, volume 13265 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2022.

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

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

[20] 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.

[21] M. Khachay and R. Dubinin. PTAS for the euclidean capacitated vehicle routing problem in rˆd. 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.

[22] 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.

[23] 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.

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

[25] C. Mathieu and H. Zhou. A ptas for capacitated vehicle routing on trees. *CoRR*, arXiv:2111.03735, 2020.

[26] C. Mathieu and H. Zhou. A tight $(\frac{3}{2} + \epsilon)$-approximation for unsplittable capacitated vehicle routing on trees. 2022.

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

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

[29] 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.

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